# Non-Strict Hierarchical Reinforcement Learning for Interactive Systems and Robots

HERIBERTO CUAYÁHUITL, Heriot-Watt University, United Kingdom
IVANA KRUIJFF-KORBAYOVÁ, German Research Centre for Artificial Intelligence, Germany
NINA DETHLEFS, Heriot-Watt University, United Kingdom

Conversational systems and robots that use reinforcement learning for policy optimization in large domains often face the problem of limited scalability. This problem has been addressed either by using function approximation techniques that estimate the approximate true value function of a policy or by using a hierarchical decomposition of a learning task into subtasks. We present a novel approach for dialogue policy optimization that combines the benefits of both hierarchical control and function approximation and that allows flexible transitions between dialogue subtasks to give human users more control over the dialogue. To this end, each reinforcement learning agent in the hierarchy is extended with a subtask transition function and a dynamic state space to allow flexible switching between subdialogues. In addition, the subtask policies are represented with linear function approximation in order to generalize the decision making to situations unseen in training. Our proposed approach is evaluated in an interactive conversational robot that learns to play quiz games. Experimental results, using simulation and real users, provide evidence that our proposed approach can lead to more flexible (natural) interactions than strict hierarchical control and that it is preferred by human users.

Categories and Subject Descriptors: I.2.6 [**Artificial Intelligence**]: Learning — Reinforcement and Supervised Learning; I.2.7 [**Artificial Intelligence**]: Natural Language Processing — Conversational Interfaces

General Terms: Algorithms, Design, Experimentation, Performance

Additional Key Words and Phrases: interactive robots, spoken dialogue systems, human-robot interaction, machine learning, reinforcement learning, hierarchical control, function approximation, user simulation, flexible interaction

**ACM Reference Format:**
Heriberto Cuayáhuitl, Ivana Kruijff-Korbayová, Nina Dethlefs, 2014. Non-Strict Hierarchical Reinforcement Learning for Flexible Interactive Systems and Robots. *ACM Trans. Interact. Intell. Syst.* 4, 3, Article 15 (October 2014), 25 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

## 1. INTRODUCTION AND MOTIVATION

There is a shared belief in the artificial intelligence community that machine learning techniques will play an important role in the development of intelligent interactive systems and robots. This is attributed to the expectation that interactive systems and robots will incorporate increasing amounts of learning skills rather than hand-coded behaviors. In this article, we focus on the application of Reinforcement Learning (RL) in order to learn behaviors from interactions in an efficient, effective and natural way. The RL framework has been an attractive and promising alternative to hand-coded policies for the design of trainable and adaptive dialogue agents. An RL agent learns its behavior from interaction with an environment and the people within it, where situations are mapped to actions by maximizing a long-term reward signal [Sutton and Barto 1998; Szepesvári 2010]. Since spoken

This work was carried out while the first author was at the German Research Center for Artificial Intelligence (DFKI GmbH) and was supported by the European Union – Integrated Project ALIZ-E (FP7-ICT-248116).
Author's addresses: Heriberto Cuayáhuitl, Heriot-Watt University, School of Mathematical and Computer Sciences, Edinbugh, UK; Ivana Kruijff-Korbayová, German Research Center for Artificial Intelligence (DFKI GmbH), Saarbrücken, Germany; and Nina Dethlefs, Heriot-Watt University, School of Mathematical and Computer Sciences, Edinbugh, UK.

DOI: http://dx.doi.org/10.1145/0000000.0000000

dialogue management was first framed as an optimization problem [Levin et al. 2000; Walker 2000; Young 2000; Singh et al. 2002], this field has experienced important progress along three main strands of work: scalability, robustness, and applicability. The first strand (*scalability*) addresses the fact that the state space growth is exponential in the number of state variables (also referred to as the *curse of dimensionality* problem). Attempts to solve this problem have involved replacing tabular representations with function approximators [Henderson et al. 2008; Li et al. 2009; Jurčíček et al. 2011], and dividing an agent into a hierarchy of agents [Cuayáhuitl et al. 2007; Cuayáhuitl et al. 2010]. The second strand (*robustness*) addresses the problem that the dialogue agent typically operates under uncertainty – the most obvious sources are speech and visual recognition errors, but they are not the only ones. Attempts to solve this problem involve finding a mapping from belief states (a probability distribution over states) to actions and have focused on keeping belief monitoring tractable by compressing the belief state [Roy et al. 2000; Williams 2007; Thomson 2009; Young et al. 2010; Crook et al. 2012]. The third strand (*applicability*) addresses interactive agents learning from real interactions. Advances have been limited here since learning algorithms usually require many dialogues to find optimal dialogue policies. Nevertheless, first attempts to solve this problem involve learning from real dialogue (for small policies) [Singh et al. 2002], learning from simulations (most of the literature), batch learning (offline) rather than online learning from interactions [Pietquin et al. 2011], fast policy learning [Gašić and Young 2011] with policy reuse [Cuayáhuitl and Dethlefs 2011], and learning from human demonstrations [Thomaz and Breazeal 2006].

While reinforcement learning dialogue systems are thus very promising, they still need to overcome several limitations to reach practical and wide-spread application in the real world. One of these limitations is the fact that user simulations need to be as realistic as possible, so that dialogue policies will not overfit the simulated interactions with poor generalization to real interactions. Another limitation is that attempts to address the curse of dimensionality often involve rule-based reductions of the state-action space [Litman et al. 2000; Singh et al. 2002; Heeman 2007; Williams 2008; Cuayáhuitl et al. 2010; Dethlefs et al. 2011]. This can lead to reduced flexibility of system behavior in terms of not letting the user take initiative in the dialogue to say and/or do anything at any time. Finally, even when function approximation techniques have been used to scale up in small-scale and single-task systems [Henderson et al. 2008; Li et al. 2009; Pietquin 2011; Jurčíček et al. 2011], their application to more complex dialogue contexts has yet to be demonstrated.

The research question that we address in this article is ***how to optimize reinforcement learning dialogue systems with multiple subdialogues for flexible human-machine interaction?*** For example, in the travel planning domain a user may wish to switch back and forth between hotel booking, flight booking and car rental subdialogues. Our motivation to aim for increased *dialogue flexibility* is the assumption that users at times deviate from the system's expected user behavior. In reduced state spaces this may lead to unseen dialogue states in which the system cannot react properly to the new situation (e.g. asking for the availability of rental cars during a hotel booking subdialogue). Thus, while a full state space represents maximal flexibility (according to the state variables taken into account), but is often not scalable, reducing the state space for increased scalability simultaneously faces the risk of reducing dialogue flexibility. Since finding the best state-action space for a learning agent is a daunting task, we suggest in this article that learning agents should optimize subdialogues and allow flexible transitions across them rather than optimizing whole dialogues. This is a novel approach that has not been explored before, but that will increase both dialogue flexibility and scalability. Our approach is couched within a Hierarchical Reinforcement Learning (HRL) framework, a principled and scalable model for optimizing sub-behaviors [Barto and Mahadevan 2003]. We extend an existing HRL algorithm with the following features:

(1) instead of imposing strict hierarchical dialogue control, we allow users to navigate more flexibly across the available subdialogues (using *non-strict hierarchical control*); and
(2) we represent the dialogue policy using function approximation in order to *generalize* the decision-making even to situations unseen in training.

Our unified approach has been evaluated using dialogues with simulated and real users. The results demonstrate that the proposed approach helps to support more flexible interactions than its non-flexible counterpart and is preferred by human users. While alternative approaches are conceivable, we argue that our approach represents a significant step towards scalable, flexible and adaptive dialogue control for interactive systems and robots.

The rest of the paper is structured as follows. We begin with a brief introduction to the Hierarchical Reinforcement Learning framework in Section 2. Section 3 will then present an illustrative example of our proposed approach and Section 4 will focus on the details of non-strict hierarchical reinforcement learning. Sections 5 and 6 describe experimental results based on simulated and real interactions with an end-to-end robot system in the quiz domain. Finally, in Section 7 we comment on related work and provide our conclusion and suggested future directions in Section 8.

## 2. BACKGROUND ON HIERARCHICAL REINFORCEMENT LEARNING

We model human-machine interaction as a sequential decision-making problem at different levels of granularity. For this purpose we define interaction control as a hierarchy of discrete Semi-Markov Decision Processes (SMDPs) in order to make scalable dialogue optimization possible. Each SMDP represents a subdialogue. A discrete-time SMDP $M = <S, A, T, R>$ is characterized by:

— a finite set of states $S = \{s_1, s_2, ..., s_{|S|}\}$;
— a finite set of actions $A = \{a_1, a_2, ..., a_{|A|}\}$;
— a stochastic state transition function $T(s', \tau|s, a)$ that specifies the next state $s'$ given the current state $s$ and action $a$, where $\tau$ is the number of time-steps taken to execute action $a$ in state $s$; and
— a reward function $R(s', \tau|s, a)$ that specifies the reward given to the agent for choosing action $a$ (lasting $\tau$ time steps) when the environment makes a transition from state $s$ to state $s'$;

SMDPs distinguish two types of actions:

(1) primitive actions (also referred to as *low-level actions*) correspond to single-step dialogue actions such as 'greeting' or 'ask question', and
(2) composite actions (also referred to as *high-level actions* or *subtasks*) are multi-step actions and correspond to subdialogues or contractions of single-step actions such as 'hotel booking' or 'car rental' in the travel planning domain.

We treat each multi-step action as a separate SMDP as described in [Cuayáhuitl 2009]. Subdialogues are executed using a stack mechanism, where the active subtask is always on top of the stack. In this way, a sequential decision-making problem can be decomposed into multiple SMDPs that are hierarchically organized into $X$ levels and $Y$ models per level, denoted as $\mu = \{M^{(i,j)}\}$,[1] where $j \in \{0, ..., X-1\}$ and $i \in \{0, ..., Y-1\}$. A given SMDP in the hierarchy is denoted as $M^{(i,j)} = <S^{(i,j)}, A^{(i,j)}, T^{(i,j)}, R^{(i,j)}>$. The solution to an optimization problem casted as an SMDP is an optimal policy $\pi^{*(i,j)}$, a mapping from dialogue states $s \in S^{(i,j)}$ to single- or multi-step actions $a \in A^{(i,j)}$. The optimal policy for each learning agent in the hierarchy is defined as

$$\pi^{*(i,j)}(s) = \arg \max_{a \in A^{(i,j)}} Q^{*(i,j)}(s, a). \tag{1}$$

For example, the HSMQ-Learning algorithm [Dietterich 2000b] approximates the $Q$-function according to the following update rule

$$NewEstimate \leftarrow OldEstimate + StepSize\left[Target - OldEstimate\right], \tag{2}$$

which, using the notation above, corresponds to

$$Q^{(i,j)}(s, a) \leftarrow Q^{(i,j)}(s, a) + \alpha \left[r + \gamma^{\tau} \max_{a'} Q^{(i,j)}(s', a') - Q^{(i,j)}(s, a)\right], \tag{3}$$

[1]The indices $i$ and $j$ only uniquely identify a subtask (SMDP) in the hierarchy, they do not specify the execution sequence of subtasks because that is learnt by the agent.

where the rewards of composite actions (lasting $\tau$ time steps) are accumulated as $r = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{\tau-1}_{t+\tau}$. Alternative learning algorithms to the one used here include MAXQ-0 [Dietterich 2000a], which can accelerate learning due to the use of a decomposed value function, or model-based approaches, such as [Cao and Ray 2012].

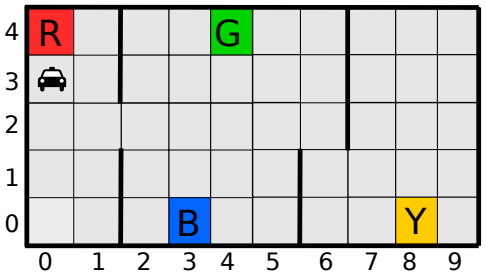
## 3. AN ILLUSTRATIVE EXAMPLE: THE INTERACTIVE TAXI

As an illustration of our proposed approach, consider the following application. A taxi has the task to bring a passenger from an origin location (R, G, B, or Y) to a destination location (R, G, B, or Y). In order to do this, the taxi needs to ask the passenger for the desired destination and it needs to collect the payment upon arrival at the destination. The taxi thus achieves its goal when the passenger has arrived at the target destination and payment has been made. This problem is an extension of the grid world taxi problem described in [Dietterich 2000c].

For illustration, we will first design a single Markov Decision Process (MDP) for the domain and then convert it into a hierarchy of SMDPs. This also allows a comparison between both models in terms of their scalability. To design an MDP for the interactive taxi, a state and action space and reward function need to be defined. The transition function can be learnt from a simulation of the domain. The sets of state variables $F = \{f_1, ..., f_n\}$ and actions $A$ are shown in Figure 1. The state transition function is based on (a) correct navigation with probability $0.8$, a random neighbour location otherwise; and (b) correct speech recognition also with probability $0.8$, a random incorrect value otherwise. The reward is as follows: +100 for reaching the goal (passenger in the destination with payment made), zero otherwise. The size of the state-action space of the interactive taxi based on a single optimization problem modeled as an MDP can be computed as follows: $|S \times A| = (\prod_{i=1}^{n} |f_i|) \times |A| = 50 \times 5 \times 5 \times 3 \times 5 \times 5 \times 4 \times 16 = 6$ million state-actions.

Let us now construct a hierarchy of SMDPs for the interactive taxi in order to make the optimization problem more scalable. The set of SMDPs, also referred to as subtasks, are shown in Table I. This table also shows the state variables and actions corresponding to each subtask. Notice that not all state variables and actions are required in each subtask. In the subtask "where" for example, when the taxi asks for the destination, it does not need to know the low-level details of how to navigate in the grid world. Sizes of the state-action spaces for each subtask in the interactive taxi are shown in the right-most column in Table I. They correspond to $|S \times A| = 2250 + 1250 + 6250 + (200 \times 4) + (80 \times 2) = 10710$ state-actions, with goal states shown in Table II. This is a much more compact search space than using flat learning which the learning agent can explore more quickly and accordingly find the optimal policy faster. Another advantage of the hierarchy is that higher-level subtasks can make joint use of lower-level subtasks, e.g. the "get" and "put" subtasks make use of the same navigation subtasks resulting in simultaneous learning.

**State Variables (features):**
taxiLoc={(0,0), (0,1), ..., (9,4)}
passengerLoc={R, G, B, Y, Taxi}
passengerDest={R, G, B, Y, unknown}
salutation={none, greeted, closed}
destination={unrequested, requested, confirmed, rejected, acknowledged}
payment={unrequested, requested, confirmed, rejected, acknowledged}
confScore={none, low, medium, high}




**Actions**={north, south, east, west, pickup, putdown, askDestination, confirmDestination, rejectDestination, acknowledgeDestination, askPayment, confirmPayment, rejectPayment, acknowledgePayment, hello, bye}

Fig. 1. Definition of the interactive taxi problem, extended from [Dietterich 2000c].

Table I. Hierarchical definition of the interactive taxi problem, where the feature set is defined in Figure 1.

| Subtask | Features | Actions | $\|S \times A\|$ |
|---|---|---|---|
| root | passengerLoc, passengerDest, payment, salutation | get, put, where, pay, hello, bye | $5^3 \times 3 \times 6$=2250 |
| get | passengerLoc, taxiLoc | nav(R), nav(B), nav(G), nav(Y), pickup | $5 \times 5 \times 50$=1250 |
| put | passengerLoc, passengerDest, taxiLoc | nav(R), nav(B), nav(G), nav(Y), putdown | $5^2 \times 50 \times 5$=6250 |
| nav(R) | taxiLoc | north, east, south, west | $50 \times 4$=200 |
| nav(B) | taxiLoc | north, east, south, west | $50 \times 4$=200 |
| nav(G) | taxiLoc | north, east, south, west | $50 \times 4$=200 |
| nav(Y) | taxiLoc | north, east, south, west | $50 \times 4$=200 |
| where | destination, confScore | askDestination, confirmDestination, rejectDestination, acknowledgeDestination | $5 \times 4 \times 4$=80 |
| pay | payment, confScore | askPayment, confirmPayment, rejectPayment, acknowledgePayment | $5 \times 4 \times 4$=80 |

Table II. Goal and terminal states per subtask in the interactive taxi problem ($*$=undefined values).

| Subtask | Goal/Terminal States | Type |
|---|---|---|
| root | passengerLoc(*) $\wedge$ passengerDes($*$) $\wedge$ payment(acknowledged) $\wedge$ salutation(closed) | goal |
| get | passengerLoc(Taxi) $\wedge$ taxiLoc(*,*) | terminal |
| put | passengerLoc($*$) $\wedge$ passengerDes($*$) $\wedge$ taxiLoc(*,*) | terminal |
| nav(R) | taxiLoc(0, 4) | terminal |
| nav(B) | taxiLoc(3, 0) | terminal |
| nav(G) | taxiLoc(4, 4) | terminal |
| nav(Y) | taxiLoc(8, 0) | terminal |
| where | destination($*$) $\wedge$ confScore($*$) | terminal |
| pay | payment(acknowledged) $\wedge$ confScore($*$) | terminal |

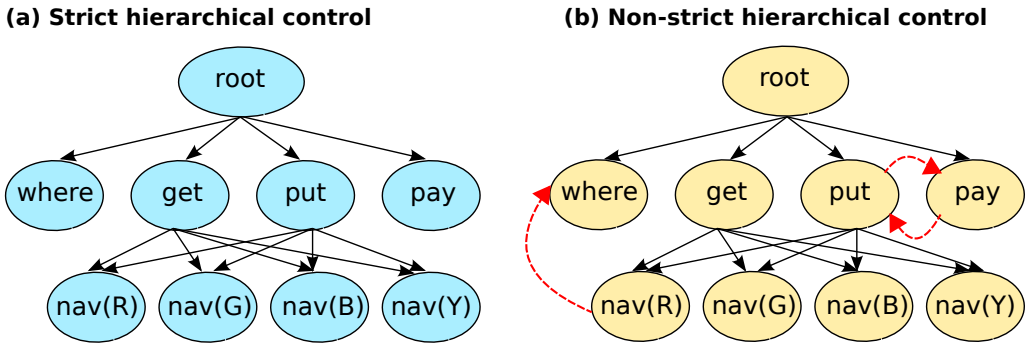



Fig. 2. Hierarchies of subtasks for the interactive taxi with strict and non-strict hierarchical control. Note that in Figure 2(b), the red arrows (i.e. between nav(R) $\rightarrow$ where and put $\leftrightarrow$ pay) are just a subset of all possible flexible transitions.

An illustration of the hierarchy of SMDPs for the interactive taxi is shown in Figure 2 (left). Here, we contrast SMDPs using strict hierarchical control (2a) with SMDPs using less-strict hierarchical control (2b). In the former case, interaction between the taxi and passenger is constrained by the structure of the SMDP. However, if a passenger wished to change their mind about their target destination in the middle of the navigation task or to pay before being put down, this is not possible using model 2a. This is because the navigation subtasks were defined to deal with navigation alone. Therefore, transitions to parent subtasks—which are able to deal with verbal requests—are forbidden in this model. Our solution is to relax the hierarchical control with more flexible transitions across subtasks, which we refer to as "non-strict hierarchical control", see Figure 2(b). The mechanism to achieve this flexibility is to extend the characterization of each SMDP in the hierarchy. An example of flexible interaction is shown in Table III, where the passenger changes their mind and requests to go to a different location. Handling such unseen situations is possible with the proposed approach described in the next section.

Table III. Sample interaction between taxi and passenger showing a change of mind in terms of the passenger's destination (line 20) in the middle of navigating to a destination. This results in the unseen (i.e. unknown) state shown in line 21, given the definition of state spaces for navigation subtasks (see Table 1). Notation: invoked subtasks in bold, user inputs in italics, and each block represents the completion of a subtask. Since a knowledge base keeps track of the state variables, we assume some initial values such as the passenger location (R), unknown destination (?), the taxi's location (0,3), among others.

| No. | Model | State | Action | Description of action |
|---|---|---|---|---|
| 01 | root | passengerLoc(R)∧passengerDes(unkown)∧ payment(unrequested)∧salutation(none) | **get** | |
| 02 | get | passengerLoc(R) | **nav(R)** | |
| 03 | nav(R) | taxiLoc(0,3) | north | |
| 04 | nav(R) | taxiLoc(0,4) | | [terminal state] |
| 05 | get | passengerLoc(R)∧taxiLoc(0,4) | pickup | |
| 06 | get | passengerLoc(Taxi)∧taxiLoc(0,4) | | [terminal state] |
| 07 | root | passengerLoc(Taxi)∧passengerDes(unkown)∧ payment(unrequested)∧salutation(none) | hello | Hi there! |
| 08 | root | passengerLoc(Taxi)∧passengerDes(unkown)∧ payment(unrequested)∧salutation(greeted) | **where** | |
| 09 | where | destination(unrequested)∧confScore(none) | askDestination | Where are you going? |
| | | | | *To location 'B' please.* |
| 10 | where | destination(requested)∧confScore(medium) | confirmDestination | Location 'B'? |
| | | | | *Yes.* |
| 11 | where | destination(confirmed)∧confScore(high) | acknowledgeDestination | Okay, to location 'B'. |
| 12 | where | destination(acknowledged)∧confScore(high) | | [terminal state] |
| 13 | root | passengerLoc(Taxi)∧passengerDes(B)∧ payment(unrequested)∧salutation(greeted) | **put** | |
| 14 | put | passengerLoc(Taxi)∧passengerDes(B) | **nav(B)** | |
| 15 | nav(B) | taxiLoc(0,4) | south | |
| 16 | nav(B) | taxiLoc(0,3) | south | |
| 17 | nav(B) | taxiLoc(0,2) | east | |
| 18 | nav(B) | taxiLoc(1,2) | east | |
| 19 | nav(B) | taxiLoc(2,2) | east | |
| 20 | | | | *Take me to 'G' instead.* |
| 21 | nav(B) | taxiLoc(3,2)∧destination(requested)∧confScore(low) | **where** | |
| 22 | where | destination(requested)∧confScore(low) | rejectDestination | Say that again. |
| | | | | *I want to go to 'G'.* |
| 23 | where | destination(requested)∧confScore(medium) | confirmDestination | No 'B' but 'G', right? |
| | | | | *Yes.* |
| 24 | where | destination(confirmed)∧confScore(high) | acknowledgeDestination | Okay, to 'G'. |
| 25 | where | destination(acknowledged)∧confScore(high) | | [terminal state] |
| 26 | nav(B) | taxiLoc(3,2) | | [interrupted subtask] |
| 27 | put | passengerLoc(Taxi)∧passengerDes(G) | | [interrupted subtask] |
| 28 | root | passengerLoc(Taxi)∧passengerDes(G)∧ payment(unrequested)∧salutation(greeted) | **put** | |
| 29 | put | passengerLoc(Taxi)∧passengerDes(G)∧taxiLoc(3,2) | **nav(G)** | |
| 30 | nav(G) | taxiLoc(3,2) | north | |
| 31 | nav(G) | taxiLoc(3,3) | north | |
| 32 | nav(G) | taxiLoc(3,4) | east | |
| 33 | nav(G) | taxiLoc(4,4) | | [terminal state] |
| 34 | put | passengerLoc(Taxi)∧passengerDes(G)∧taxiLoc(4,4) | putdown | |
| 35 | put | passengerLoc(G)∧passengerDes(G)∧taxiLoc(4,4) | | [terminal state] |
| 36 | root | passengerLoc(G)∧passengerDes(G)∧ payment(unrequested)∧salutation(greeted) | **pay** | |
| 37 | pay | payment(unrequested)∧confScore(none) | askPayment | The fare is £7.50 |
| | | | | *Seven fifty?* |
| 38 | pay | payment(requested)∧confScore(high) | confirmPayment | That's right. |
| | | | | *There you go.* |
| 39 | pay | payment(confirmed)∧confScore(high) | acknowledgePayment | Thank you! |
| 40 | pay | payment(acknowledged)∧confScore(high) | | [terminal state] |
| 41 | root | passengerLoc(G)∧passengerDes(G)∧ payment(acknowledged)∧salutation(greeted) | bye | Good bye! |
| 42 | root | passengerLoc(G)∧passengerDes(G)∧ payment(acknowledged)∧salutation(closed) | | [goal state] |

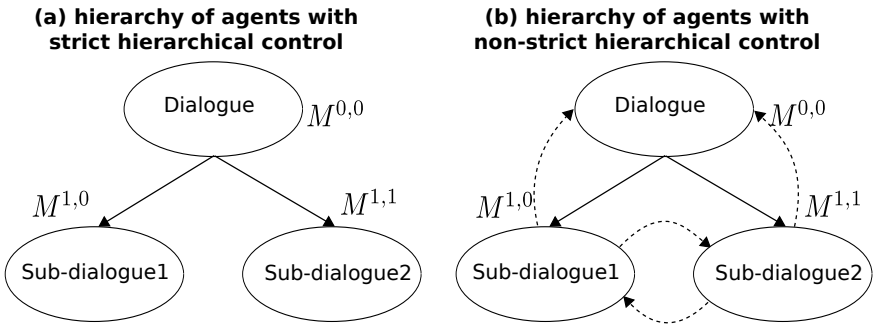



Fig. 3. Hierarchies of agents with strict and non-strict hierarchical execution. While the straight arrows connecting models $M^{i,j}$ mean invoking a child model and returning control after terminating its execution, the dashed arrows connecting models mean interrupting the execution of the current model and transition to another model to continue the interaction.

## 4. PROPOSED APPROACH

### 4.1. Non-Strict Hierarchical Reinforcement Learning

This section proposes an extension of the hierarchical RL framework described above in order to support scalable and flexible interaction policy learning. Rather than using traditional strict hierarchical control, we propose to use a less-strict hierarchical control by allowing transitions between dialogue subtasks. In addition, we represent the dialogue policies using linear function approximation. In this way, the learnt policy can make (more generalized) decisions even for unseen states. Figure 3 illustrates the difference between our approach (3b) and a strict hierarchical control (3a). While in the strict setting, only transitions within subtasks are allowed, our proposed model allows all possible transitions except for self-transitions (to avoid loops). The non-strict hierarchical control achieved in this way therefore allows users to act freely at anytime and across subdialogues.

To achieve more flexible navigation across subdialogues, we extend the previous formulation of SMDP models as $M^{(i,j)} = <S^{(i,j)}, A^{(i,j)}, T^{(i,j)}, R^{(i,j)}, G^{(i,j)}>$, where the newly added element $G^{(i,j)} = P(m'|m, s, a)$ is a probabilistic **subtask transition function** that specifies the next subtask $m' \in \mu$ in focus given the current subtask $m$, state $s$ and action $a$. While $m$ refers to a subtask in the set of subtasks $\{M^{(i,j)}\}$, $G^{(i,j)}$ represents the mechanism to specify the currently active subtask. In addition, the presence of unseen situations—due to the factorization of the state action space of each subtask $M^{(i,j)}$—involves **dynamic state spaces** that add unseen states to their state spaces $S^{(i,j)}$. This implies a growth from $S_t^{(i,j)}$ to $S_{t+1}^{(i,j)} = \{S_t^{(i,j)} \cup \bar{s}\}$, where $\bar{s}$ represents the unseen state. Let us look at the example interaction shown in Table III, where the passenger changes the destination when subtask $nav(B)$ is being executed (see line 20). This event results in the unseen state $\bar{s} =$ taxiLoc(3,2) $\wedge$ destination(requested) $\wedge$ confScore(low), where $\bar{s} \notin S^{nav(B)}$ (line 21), derived by a knowledge base update mechanism (see below in this section). The presence of unseen situations requires some form of **function approximation** to make decisions for unknown state-action pairs. We propose to use linear function approximation, though other function approximators are possible. The policies for these extended SMDPs behave according to

$$\pi_\theta^{*(i,j)}(s) = \arg \max_{a \in A^{(i,j)}} Q_\theta^{*(i,j)}(s, a), \tag{4}$$

where the Q-funcion is represented by a set of weighted linear functions expressed as

$$Q_\theta^{(i,j)}(s, a) = \theta_0^{(i,j)} + \theta_1^{(i,j)}\phi_1(s, a) + ... + \theta_n^{(i,j)}\phi_n(s, a) \tag{5}$$

with a set of feature functions $\Phi = \{\phi_1, ..., \phi_n\}$ and parameters $\theta^{(i,j)} = \{\theta_0^{(i,j)}, ...\theta_n^{(i,j)}\}$ for each agent in the hierarchy. We assume binary feature functions derived from joint features and actions.

An example feature function in our interactive taxi is as follows:

$$\phi_1(s,a) = \begin{cases} 1 \text{ if destination=unrequested} \in s \wedge a\text{=askDestination} \\ 0 \text{ otherwise.} \end{cases} \quad (6)$$

A reinforcement learning agent can learn values for the parameters $\theta$. For example, HSMQ-Learning with linear function approximation estimates parameters according to

$$\theta^{(i,j)} \leftarrow \theta^{(i,j)} + \alpha \left[ r + \gamma^{\tau} max_{a' \in A^{(i,j)}} Q_{\theta}^{(i,j)}(s',a') - Q_{\theta}^{(i,j)}(s,a) \right] \Phi(s,a), \quad (7)$$

and other algorithms can be extended in a similar way. The *FlexHSMQ-Learning* algorithm shown below simultaneously learns a hierarchy of action-value functions. It is based on the HSMQ-Learning algorithm originally proposed by [Dietterich 2000b] with strict hierarchical control. This **learning algorithm** receives subtask $M^{(i,j)}$, and **knowledge base** $K$ used to initialize state $s$. The knowledge base keeps track of all the information of the interaction history through discrete random variables, updated after each executed action (primitive or composite). Its role is to facilitate the storage and retrieval of random variables, and to inform subtasks about active random variables. Note that handling such random variables implies maintaining domain values for each variable (e.g. the names of locations in a navigation environment) and maintaining values of observed variables (e.g. in our taxi example, whether the passenger's destination has been requested or not). While the former are used to maintain a rich state of the environment (required by real human-machine interactions), the latter are used to learn the agents' behavior based on compact representations.

Our algorithm—independently of how $K$ is modelled—performs similarly to Q-Learning for primitive actions, but for composite actions it invokes recursively with a child subtask. The original execution of subtasks uses a **stack of SMDPs** and operates as follows: the dialogue starts with the active root subtask $M^{(0,0)}$ on the stack; when a child subtask $M^{(1,j)}$ is selected, it is pushed onto the stack and control is transferred to the child subtask which is executed until reaching a terminal state—this may involve a recursive execution of other subtasks that may reach the bottom of the hierarchy; then the current subtask is popped off the stack and control is transferred back to the parent subtask at the next state $s' \in S^{(i,j)}$; this process continues until the execution of the root subtask is completed, which empties the stack and terminates the dialogue. When a given subtask is executed with $\tau$ time steps, it returns a cumulative reward $r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{\tau-1} r_{t+\tau}$, and the RL agent continues its execution until finding a goal state for the root subtask $M^{(0,0)}$.

If during the execution of a subtask the user decides to jump to another subtask (see Table III, lines 20-21), the flexible execution of subtasks proposed here allows each subtask to be interrupted in two ways in order to transition to another subtask (this process is formalized in Algorithm 1):

— First, we check whether the new (active) subtask is already on the stack of subtasks to execute. This would be the case if it were a parent/ancestor of the current subtask. In this case, we interrupt execution of all intervening subtasks until the new active subtask is on top of the stack. Notice that the interruption of all intervening subtasks prevents the stack from growing infinitely. As an example of this form of transition in the interactive taxi domain, assume that the taxi is navigating to location $G$ when the passenger suddenly asks to be put down in the current location instead. This would cause the interruption of subtask *nav(G)* and then transfer control to the parent subtask *put* to execute the desired action (then the payment is made and the interaction is closed).

— Second, if the new active subtask is not already on the stack of subtasks to execute, it is pushed onto the stack and control is passed to it. Once the new subtask terminates execution, control is transferred back to the root subtask. Notice that transferring control back to the root subtask after an interrupted subtask makes the interaction consistent with the given hierarchy. Thus, transferring control to the root is a safe move because the interaction would continue according to the current state of the world (as specified by the knowledge base). In our taxi scenario, if the passenger requests a different destination in the middle of a navigation task, we do not want to

**Algorithm 1** Flexible HSMQ-Learning

```
1:  function FLEXHSMQ(KnowledgeBase K, Stack 𝕊) return totalReward
2:      m ← current subtask M^(i,j), i.e. the subtask on top of 𝕊
3:      s ← environment state in S^m initialized from K
4:      totalReward ← 0, discount ← 1
5:      while s is not a terminal or goal state and subtask on top of 𝕊 is active do
6:          Choose action a ∈ A^m from s using policy derived from Q^m (e.g. ε-greedy)
7:          Execute action a and update knowledge base K
8:          Observe resulting subtask m′ (from subtask transition function G^m)
9:          if m <> m′ and m′ ∈ 𝒮 then                          ▷ Flexible Transition: Case 1
10:             Set m′ as the new active subtask
11:         else if m <> m′ and m′ ∉ 𝒮 then                     ▷ Flexible Transition: Case 2
12:             Set m′ as the new active subtask and push it onto 𝕊
13:             r ← FlexHSMQ(K, 𝕊)
14:             Set the root subtask as the new active subtask
15:         else
16:             if a is primitive then
17:                 r ← Observed one-step reward (from reward function R^m)
18:             else if a is composite then                     ▷ Regular Subtask Transition
19:                 Set a as the new active subtask and push it onto 𝕊
20:                 r ← FlexHSMQ(K, 𝕊)
21:             end if
22:             totalReward ← totalReward + discount × r
23:             discount ← discount × γ
24:             Observe resulting state s′ (from state transition function T^m)
25:             Update rule, e.g. equation 7 (linear function approximation)
26:             s ← s′
27:         end if
28:         m ← m′
29:     end while
30:     Pop subtask on top of 𝕊
31: end function
```

return to the previous navigation task. Instead, going to the root agent after requesting the new goal location results in the taxi choosing more appropriate subtasks for a successful navigation. Figure 4 shows this form of transition in the interactive taxi. It can be observed that the change of destination in the middle of a navigation task (see Table III, lines 20-21) causes the observed subtask *where* to be pushed onto the stack (stack operation 10 in Figure 4), and then control is transferred to the root subtask (stack operation 11).

While transitions involved in switching subtasks can be seen as high-level transitions in the entire state space, transitions within subtasks can be seen as low-level transitions in a region of the entire state space. We therefore maintain one active subtask at each point in the interaction (through the subtask transition function, see line 8 in Algorithm 1) for the high-level transitions, and the observed state (through the dynamic state space) for the low-level transitions. The dialogue history is maintained in the knowledge base $K$ and therefore the initial states of each subtask are initialized accordingly. Notice that the mechanism used to update the knowledge base is independent of the learning algorithm. Since the learning algorithm executes actions in a top-down fashion (based on the stack mechanism), even in the presence of interrupted subtasks, this algorithm maintains a hierarchical execution despite the flexible transitions.

### 4.2. Applying the Proposed Approach to Any Domain

The concepts and algorithm above are brought together according to the following methodology:

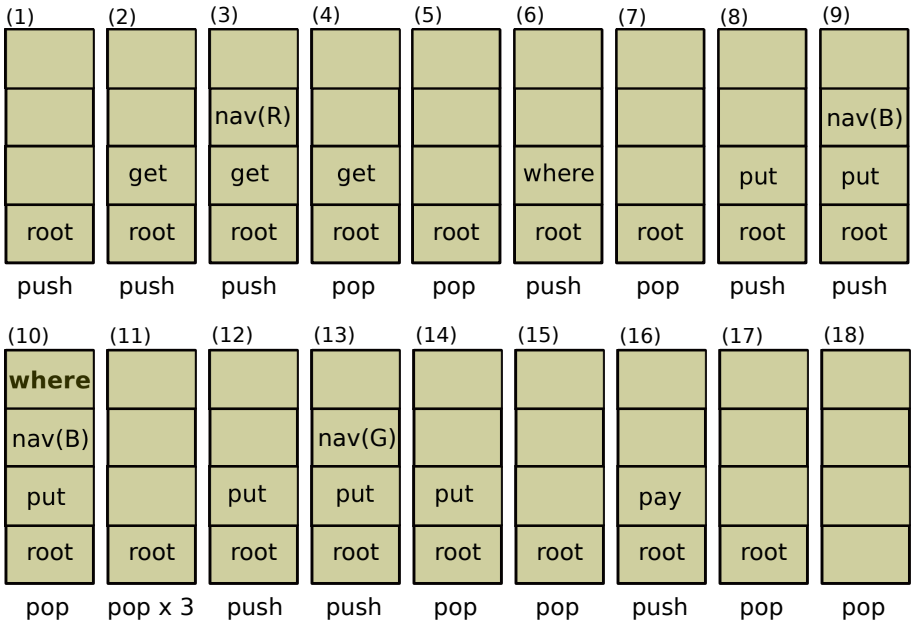



Fig. 4. Stack operations as part of the example in Table III. Briefly, the Taxi navigates to location $R$ to pick up the passenger (operation 3). It asks for the destination (operation 6), it navigates to the destination (operation 9), the user changes the destination (operation 10), the taxi navigates to the new destination (operation 13), the interaction terminates (operation 18).

(1) Define the set of subtasks $\mu = \{M^{(i,j)}\}$ and their dependencies (hierarchy), which we assume to be manually provided but it can be induced automatically (we leave this as future work);
(2) Define a set of state variables $f_i$ to characterize the state space $S = \{s_1, ..., s_n\}$ of each subtask;
(3) Define the set of actions $A = \{a_1, ..., a_m\}$ for each subtask and constraints if they are known;
(4) Define or learn the state transition function $T = P(s'|s, a)$ for each subtask in the hierarchy, e.g. by using graphical models trained from example interactions;
(5) Define or learn the subtask transition function $G = P(m'|m, s, a)$ for each subtask in the hierarchy, for example by using graphical models trained from example interactions (e.g. Wizard-Of-Oz data). This function is a prior requirement not learnt by the proposed algorithm above;
(6) Define knowledge base $K$ to facilitate the storage and retrieval of random variables, and to inform subtasks about active random variables.
(7) Define the reward function $R(s'|s, a)$ for the learning agents, where this function can also be induced, for example, by using regression methods (we also leave this as future work);
(8) Train action-value functions $Q_\theta(s, a)$, for example by using simulated interactions and a learning algorithm with support for flexible transitions across subtasks such as Algorithm 1; and
(9) Test the learned policies $\pi^*_\theta(s)$ in a real environment with human subjects.

The reader should notice that this methodology is independent of the learning algorithm as long as it offers support for flexible transitions across subtasks. In the rest of the article we describe an application of non-strict hierarchical RL to a human-robot interaction domain.

## 5. SIMULATION-BASED EVALUATION

To test whether our approach can lead to more flexible interactions even in a more complex setting than the interactive taxi domain, this section will report some simulation-based results in a domain with a humanoid robot that plays a quiz game with a human interlocutor. In this scenario, the robot and user can ask each other questions about a variety of topics. Usually, one player will take the role

of asking questions until roles are switched. In addition, to allow flexibility across dialogue subtasks, both user and robot can switch roles or stop playing at any point during the interaction, i.e. from any dialogue state. In particular, this section serves to compare the HSMQ-Learning algorithm (our baseline) against FlexHSMQ-Learning (our proposed algorithm).

### 5.1. Characterization of the Learning Agents

To learn a policy, we use a small **hierarchy** of dialogue agents with one parent and two children agents ('robot asks' and 'user asks'), which is equivalent to the hierarchy shown in Figure 3. Table IV shows the set of state variables for our system. While the **state space** of the parent (root) agent makes use of the discrete state variables $\{f1, ..., f_{11}, f_{17}\}$, the state spaces of the children agents ('robot asks' and 'user asks') make use of the state variables $\{f_4, f_6, f_{11}, ..., f_{16}, f_{18}\}$. Although our set of **actions** consists of over 60 meaningful combinations of speech act types[2] and associated parameters[3] as shown in Table V, we used prior knowledge to constrain the possible actions at each point. For example: action *Salutation(Greet)* is valid in state $s$ if feature-value pair $f_2 = none \in s$. This resulted in an average search space branching factor of 3.1 actions per state. Notice that enumerating all state variables and values leads to over $|S \times A| = 10^{12}$ state-action pairs. This makes a combination of hierarchical RL and function approximation attractive, not only to scale up, but in order to solve the problem using multiple unified solutions and to optimize subdialogues rather than one whole long dialogue.

We represent the **state transition functions** as a Bayesian network with the set of discrete random variables $\{f_i, f_i', a\}$, where $f_i'$ is the feature set $f_i$ at time step $t+1$. The structure and parameters of the network were estimated from Wizard-of-Oz data (21 interactions) [Belpaeme et al. 2012]. For probabilistic inference we used the junction tree algorithm. This enabled us to pose queries such as $P(f_1'|f_1{=}val(f_1), ..., f_n{=}val(f_n), a{=}val(a))$, where $\{f_i\}$ is the set of features that describe the last environment state, and $a$ is the last executed action. Such probabilistic queries can be posed for each feature $f_i'$ representing the class label. The feature values derived from probabilistic queries are communicated to the knowledge base $K$.

Similarly, we represent the **subtask transition functions** as a separate Bayesian network with the set of discrete random variables $\{f_i, a, m, m'\}$, where subtask $m'$ is the class label. The training and inference procedure are similar to the Bayesian net above. This enabled us to pose queries such as $P(m'|m{=}val(m), f_1{=}val(f_1), ..., f_n{=}val(f_n), a{=}val(a))$, where $m'$ is the next subtask, $m$ is the current subtask, $\{f_i\}$ is the set of the features that describe the most recent environment state and $a$ is the last executed action. The feature values derived from these probabilistic queries are also communicated to the knowledge base $K$.

Furthermore, the **reward function** prefers interactions with continued play and getting the right answers. It is defined by the following rewards for choosing action $a$ in state $s$:

$$r = \begin{cases} \text{+10 for reaching a terminal/goal state or answering a question correctly,} \\ \text{-10 for remaining in the same state,} \\ \text{0 otherwise.} \end{cases} \quad (8)$$

The **user simulation** used a set of user dialogue acts equivalent to the system actions (except for composite actions). The user acts or responses were sampled from bigram language models $P(a^{usr}|a^{sys}{=}val(a^{sys}))$ with Witten-Bell discounting from 21 wizarded dialogues (900 user turns). The values of user responses were distorted based on an equally distributed speech recognition error rate of 20%. The recognition confidence scores were generated from beta probability distributions with parameters ($\alpha$=2, $\beta$=6) for bad recognition and ($\alpha$=6, $\beta$=2) for good recognition.

[2]Set of speech act types: Salutation, Request, Apology, Confirm, Accept, SwitchRole, Acknowledgement, Provide, Stop, Feedback. Express, Classify, Retrieve, Provide.

[3]Parameters of speech act types: Greet, Closing, Name, PlayGame, Asker, KeepPlaying, GameFun, StopPlaying, Play, NoPlay, Fun, NoFun, GameInstructions, StartGame, Question, Answers, CorrectAnswer, IncorrectAnswer, GamePerformance, Answer, Success, Failure, GlobalGameScore, ContinuePlaying.

Table IV. Discretized state variables for the reinforcement learning dialogue agents in the quiz domain.

| ID | State Variable | Values |
|---|---|---|
| $f_1$ | Quiz | root, UserAsks, RobotAsks |
| $f_2$ | Salutation | null, none, greeting, withName, regreeting, closing |
| $f_3$ | UserName | null, unknown, filled, known |
| $f_4$ | ConfScore | null, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 |
| $f_5$ | Confirmed | null, no, yes |
| $f_6$ | PlayGame | null, unknown, no, yes, ready |
| $f_7$ | Instructions | null, unprovided, provided |
| $f_8$ | Asker | null, unknown, robot, user |
| $f_9$ | QuizGame | null, unplayed, playing, semiplayed, played, interrupted, keepPlaying, stopPlaying |
| $f_{10}$ | GameFun | null, unknown, no, yes |
| $f_{11}$ | GameOver | null, no, yes |
| $f_{12}$ | GameInstructions | null, unprovided, provided |
| $f_{13}$ | QuestionState | null, unknown, unasked, confirmed, askedWithoutAnswers, askedWithAnswers, reaskedWithoutAnswers, reaskedWithAnswers, reaskedWithoutAcknowledgement, askedButHeardBefore, askedButUnheardBefore, askedWithUnknownQuestion |
| $f_{14}$ | AnswerState | null, unanswered, unclassified, correct, incorrect, unknown, revealed |
| $f_{15}$ | MaxQuestions | null, no, yes |
| $f_{16}$ | GameScore | null, unknown, good, bad |
| $f_{17}$ | GlobalGameScore | null, unprovided, semi-provided, provided |
| $f_{18}$ | ExpressedScore | null, no, yes |

### 5.2. Simulation-Based Results

We trained our agents using strict hierarchical control (this is our baseline) and our proposed approach, and compared their performance in terms of *dialogue reward*, see Figure 5 (left). While the baseline used the HSMQ-Learning algorithm, our approach used the FlexHSMQ-Learning algorithm; both used linear function approximation. Results show that the proposed approach, using flexible dialogue control, clearly outperformed its counterpart using strict-hierarchical control. The reason for this is the more flexible interaction behavior displayed by the non-strict control learner. Since this agent was able to support transitions across subtasks, it received less negative rewards for failing to make a transition when the user requested one. For example, even when a simulated user had said 'stop playing' previously, they could revert this choice and continue playing. This is in contrast to the strict-control learner, which was unable to support "topic changes" initiated by the user. This suggests that dialogue agents with more flexible dialogue structures are more likely to be successful than systems with fixed and rigid dialogue structures.

In addition, we kept track of an initially specified state space (see Appendix A) in order to observe its growth due to unseen situations during policy training. Figure 5 (right) shows our system's state-action space growth, where the flexible system grew by 29 times over the non-flexible one. This shows that the user simulations made use of subdialogues at different points during the interaction, suggesting an enriched dialogue structure flexibility. Since simulated interactions can only provide first indications of the compared performance of policies, the next section will confirm our results in a study with human users. To this end, the non-strict dialogue policies have been integrated into an end-to-end robot dialogue system.

## 6. EVALUATION WITH HUMAN USERS

In this section, we will evaluate the interaction policies learnt in simulation (see previous section) by testing them in interaction with real users and a physical humanoid robot. It has to be noted that the policies were deployed with frozen learning, i.e. no learning while interacting with humans.

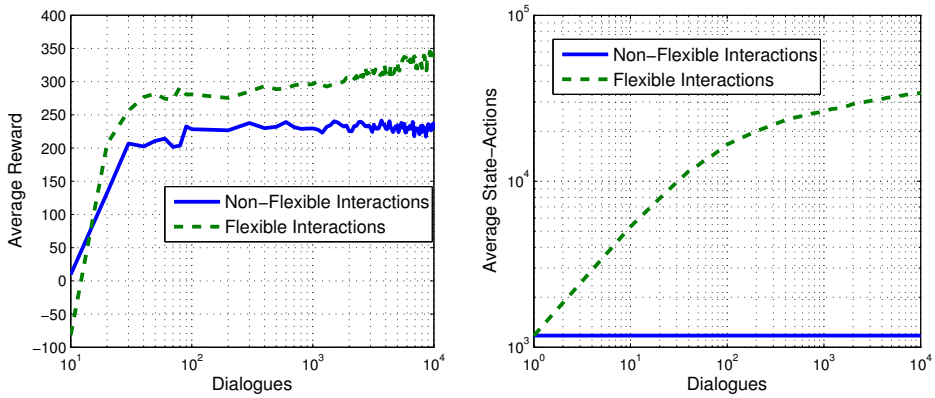



Fig. 5. Average (10 runs) dialogue reward (left) and state growth (right) of our robot dialogue system.

### 6.1. The Robot Dialogue System

Our experiments were carried out using the human-robot interaction system developed in the EU-funded ALIZ-E project[4] using the Nao robot.[5] The system integrates components for speech and gesture capture and interpretation, activity and interaction management, user modeling, speech and gesture production and robot motor control (see Figure 6). We use components developed within the project as well as off-the-shelf technologies such as Google speech recognition, OpenCV[6] for gesture recognition, Acapela[7] for speech synthesis, OpenCCG[8] for language parsing and generation, Weka[9] and JavaBayes[10] for representing the subtask transition functions and for maintaining a probabilistic personalized user profile. To bring all components together within a concurrent execution approach, we use the Urbi middleware [Baillie 2005]. More details on the system implementation are described in [Kruijff-Korbayová et al. 2012a; 2012b]. In the experiments described in this article, we focused on speech-based interactions and omitted gesture recognition; though see [Cuayáhuitl and Kruijff-Korbayová 2011] for dialogue policy learning combining speech and visual inputs. Table V shows a sample interaction between a user and the robot.

During interactions, the users provided spoken input through a smartphone, which were processed by a Bayes Net dialogue act recogniser according to $a^{usr} = \arg\max_{a^{usr}} P(a^{usr}|evidence)$, where $a^{usr}$ represents the spoken user response and $evidence = \{f_1 = val_1, ..., f_n = val_n\}$ is a set of contextual feature-value pairs[11]. This Bayesian classifier was trained in a supervised learning manner from the same wizarded interactions mentioned in the previous section. Based on this input, the knowledge base is updated and the next system action is selected by the Interaction Manager (dialogue policies) trained as described above. The dialogue act corresponding to the selected next system action is verbalized automatically by the Natural Language Generation component which produces text for the speech synthesizer. Subsequently, nonverbal behavior planning and motor control (i.e. automatic communicative gestures) are evoked to accompany specific types of dialogue

[4]project website http://aliz-e.org
[5]http://www.aldebaran-robotics.com
[6]http://opencv.willowgarage.com/wiki/
[7]http://www.acapela-group.com/
[8]http://openccg.sourceforge.net/
[9]http://www.cs.waikato.ac.nz/ml/weka/
[10]http://www.cs.cmu.edu/~javabayes/Home/
[11]Features for the Bayes net dialogue act recogniser: lastSystemDialogueAct, subtask, hasName, hasYes, hasNo, hasCorrect, hasIncorrect, hasRepeat, hasStop, hasContinue, hasAskMe, hasAskYou, hasAnswer, hasAnswers, hasIDontKnow, hasQuestion; where the first two are non-binary features and the rest are binary ones filled from speech recognition N-best hypotheses.

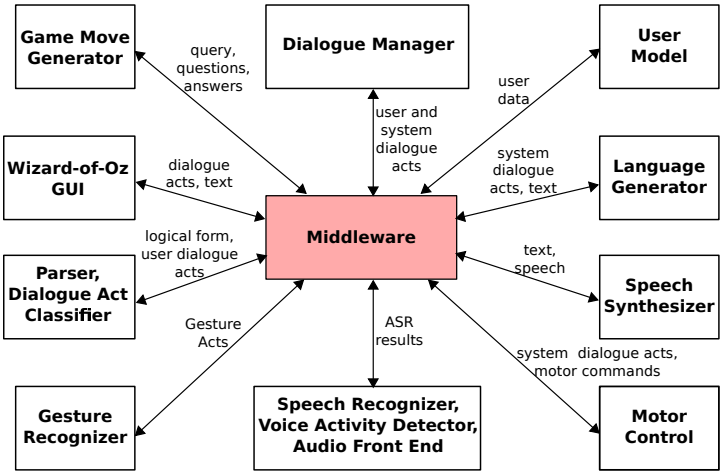



Fig. 6. High-level architecture of the integrated robot system.

acts (e.g., greetings, requests) as well as static key poses that display emotions such as anger, sadness, fear, happiness, excitement and pride [Beck et al. 2010]. The following features summarize the capabilities of the interactive conversational robot used in our experiments:

(1) automatic speech and dialogue act recognition;
(2) push to talk through a mobile phone;
(3) automatic system action selection;
(4) user barge-in: interruption of the robot's speech by an early user response;
(5) automatically produced verbal output in English with many variations and expressive speech synthesis distinguishing sad, happy and neutral state;
(6) automatically produced head and body poses and gestures;
(7) random motion of the upper half body to give the impression that the robot is alive (also known as "perlin noise"); and
(8) persistent user-specific interaction profile, so that subsequent interactions would take into account what is known about a particular user in the current game (such as the questions asked and answered, role switches, etc.)

### 6.2. Experimental Methodology and Setup

Thirty-seven participants with an average age of 26.7, who were mostly master degree students from different fields, took part in our evaluation. Our participants were fluent speakers of English. Figure 7 shows our experimental setup and several users interacting with the robot. Each user was presented with two tasks—one with the baseline system and one with the flexible one—where each involved playing the quiz game in English with our interactive autonomous robot for approximately 15 minutes (i.e. 30 minutes in total per participant). While the *flexible system* included non-strict hierarchical control and dynamic states, the *baseline system* used strict hierarchical control and static states. The goal was to answer as many questions as possible and to ask as many questions as possible from a set of given questions (130 in total from different fields including science, literature, history, geography, health, art, music, sports and cinema). The robot's default high-level strategy was to play rounds of 3-questions per conversant with role switching after 3-questions. While half of the participants started with the strict system, the other half started with the flexible system to avoid ordering effects. In contrast to wizarded interactions (those that served as training data) where users were not explicitly encouraged to switch roles, in real interactions users were told that they

Table V. Sample interaction, where user utterances are shown in italics and star symbols omit parameter values.

| Agent | Dialogue Action | Utterance | Subtask |
|---|---|---|---|
| Robot | Salutation(Greeting) | Hi, my name is NAO. What's your name? | $M^{(0,0)}$ |
| User | Provide(Name=Harry) | *I am Harry.* | |
| Robot | Salutation(withName, Name=Harry) | Nice to meet you Harry. | |
| | Request(PlayGame, Game=Quiz) | Do you want to Play a quiz game? | |
| User | Confirm(PlayGame=yes) | *Sure.* | |
| Robot | Provide(Instructions) | One of us asks questions and offers possible answers. | |
| | | The other answers by selecting one of those answers. | |
| | Request(Asker) | Do you want to ask me questions or shall I ask you? | |
| User | Provide(Asker=robot) | *You ask me.* | |
| Robot | SwitchRole(Asker=Robot) | Okay, I'll ask you. | $M^{(1,0)}$ |
| | Retrieve(Question)+Provide(Question*) | The question is: What is a typical magic instrument? | |
| | Provide(Answers*) | Your choices are: 1 guitar, 2 drums, 3 violin, 4 flute. | |
| | Request(Answer) | What is your choice? | |
| User | Provide(Answer=drums) | *Number two.* | |
| Robot | Feedback(IncorrectAnswer) | That is not quite right. | |
| | Request(AnswerAgain) | Do you have another choice? | |
| User | Provide(Answer=violin) | *Maybe number three?* | |
| Robot | Provide(CorrectAnswer=flute) | The correct answer is number four, flute. | |
| | Retrieve(Question)+Provide(Question*) | The next question is: | |
| | | Who invented the steam engine? Your choices are: | |
| | Provide(Answers*) | 1 Marconi, 2 James Watt, 3 Tom Savery, 4 The Wright Bros. | |
| | Request(Answer) | What is your choice? | |
| User | Provide(Answer=James Watt) | *James Watt.* | |
| Robot | Feedback(CorrectAnswer) | Yes, correct. | |
| | Retrieve(Question)+Provide(Question*) | The next one is: Which are the elements of a football goal? | |
| | Provide(Answers*) | Your choices are: 1 poles, cross and net, | |
| | | 2 cross, net and arbiter, 3 poles, ball and grass. | |
| | Request(Answer) | What is your choice? | |
| User | Provide(Answer=poles, cross and net) | *The first one.* | |
| Robot | Feedback(CorrectAnswer) | Yes, correct. | |
| | Provide(GamePerformance*) | Well done, you got two questions out of three right. | |
| Robot | SwitchRoleTo(user) | Okay, let's switch roles. Now you ask me. | $M^{(0,0)}$ |
| Robot | Request(Question) | What is the first question? | $M^{(1,1)}$ |
| User | Provide(Question*) | *What does water become at the temperature of zero degrees?* | |
| Robot | Request(Answers) | What are my choices? | |
| User | Provide(Answers) | *Your choices are: 1 condensation, 2 ice, 3 steam.* | |
| Robot | Provide(Answer=condensation) | I choose condensation. | |
| User | Feedback(IncorrectAnswer) | *That is incorrect.* | |
| Robot | Express(Failure)(IncorrectAnswer) | Oh no. | |
| | Request(CorrectAnswer) | So what is the right answer? | |
| User | Provide(CorrectAnswer=ice) | *The correct answer is number two.* | |
| Robot | Request(Question) | Okay, ask me another question. | |
| User | Provide(Question*) | *The next question is: Through which breach* | |
| | | *did the Italian army enter into Rome in 1870?* | |
| Robot | Request(Answers) | What are my choices? | |
| User | Provide(Answers) | *1 Porta Venezia, 2 Porta Pia, 3 Porta Pazienza, 4 Porta Nuova.* | |
| | Request(Answer) | *What is your choice?* | |
| Robot | Provide(Answer=Porta Pia) | Porta Pia. | |
| User | Feedback(CorrectAnswer) | *Yes, that's right.* | |
| | Provide(Question*) | *The next question is: Which city has the largest population?* | |
| User | Provide(Answers*) | *Choices: 1 Shanghai, 2 Rio de Janeiro, 3 New York, 4 Tokyo.* | |
| | Request(Answer) | *What is your choice?* | |
| Robot | Provide(Answer=we) | I think it is Tokyo. | |
| User | Feedback(CorrectAnswer) | *Yes, correct.* | |
| Robot | Provide(GamePerformance*) | Not bad, I got two questions out of three right. | |
| | Request(KeepPlaying) | Shall we keep playing? | $M^{(0,0)}$ |
| User | Provide(QuizGame=stopPlaying) | *I want to stop.* | |
| Robot | Provide(GlobalGameScore*) | In this session you got right 2 out of 3 questions. | |
| | Request(GameFun) | Did you like playing the quiz Game? | |
| User | Provide(GameFun=yes) | *I did.* | |
| Robot | Acknowledgement(Fun) | I am glad to hear that. | |
| | Salutation(closing) | It was nice playing with you, see you soon. Bye! | |

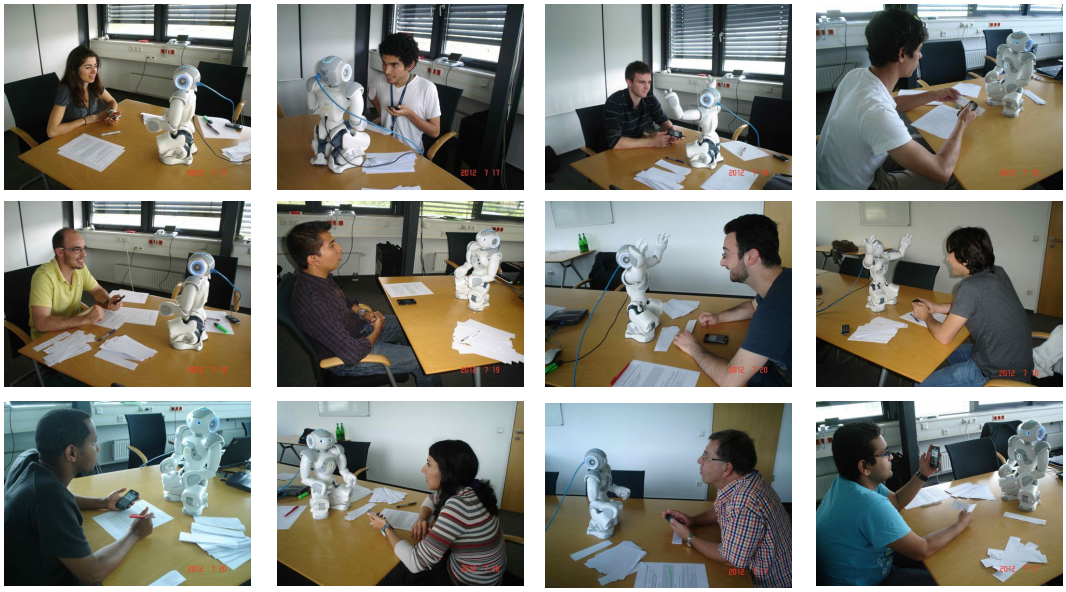


Fig. 7. Some of our participants talking to the Nao robot while playing the quiz game. The pieces of paper on the table are used by the user to ask questions and to provide perceived performance through a questionnaire (given at the end of each ∼15 minute interaction). All participants interacted with our two proposed systems. While half of the participants started playing with the baseline system, the other half started playing with the flexible one.

could decide who asks and who responds rather than just obeying the robot. This was the motivation to increase user initiative by challenging the dialogue structure imposed by the robot. Users were given these instructions at the beginning of every session:

(1) speak naturally whenever interacting with the robot,
(2) you can say 'I don't know' if you do not know the answer to a question,
(3) you can select any of the questions that you will be given on small pieces of paper,
(4) you will be told when your time runs out so that you can stop playing,
(5) you can decide when to ask questions and when the robot should ask.

At the end of each task, users were asked to fill the questionnaire shown in Table 8 (column 1) for obtaining qualitative results using a 5-point Likert scale, where 5 represents the highest score.

### 6.3. Results with Real Users

Qualitative results are shown in Table VI and as a bar plot in Figure 8. These results show that the policy with more flexibility outperformed the non-flexible one in almost all metrics except Q7 (*robot speed*; see Table VI). These differences are significant for metrics Q1 (*easy to understand*), Q2 (*robot understood*), Q5 (*interaction pace*) and Q8 (*expected behavior*). All of these metrics can be said to measure the flexibility in the robot's interaction behavior in one way or another. For example, if the robot does not switch the subtask when the users request it, this could be interpreted as the robot not understanding. Also, users generally felt that the robot using flexible dialogue control corresponded to their expectations better, again for example, switching the topic when requested. The overall comparison of behaviors per participant (most right column in Figure 8) showed that the flexible policy outperformed its counterpart by 4.75% at $p = 0.0002$, based on a two-sided Wilcoxon Signed-Rank test (paired difference test). Based on these results, we can conclude that enhancing the flexibility of an interactive robot—by allowing non-strict hierarchical navigation across subdialogues using dynamic states—can significantly improve user satisfaction.

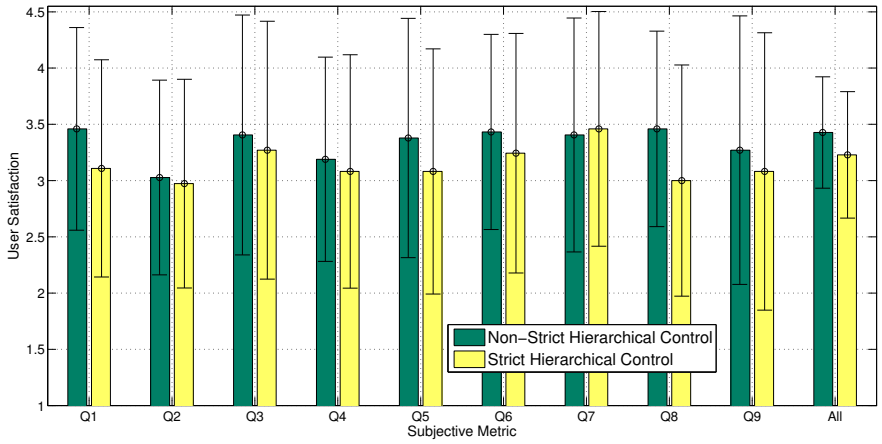



Fig. 8. Average subjective results of the robot interactive system in the quiz domain (overall in the two most-right bars).

Table VI. User satisfaction results for the subjective metrics for strict and non-strict hierarchical control. Numbers refer to averages and are shown together with their standard deviations. The overall difference between both systems is significant at $p < 0.0002$.

| **Subjective Metric** | **Flexible** | **Non-Flexible** | $p$**-value** |
|---|---|---|---|
| (Q1) Was the robot easy to understand? | $3.46 \pm 0.9$ | $3.11 \pm 0.97$ | **0.01** |
| (Q2) Did the robot understand what you said? | $3.03 \pm 0.85$ | $2.97 \pm 0.93$ | **0.01** |
| (Q3) Was it easy to ask questions? | $3.41 \pm 1.06$ | $3.27 \pm 1.15$ | 0.2 |
| (Q4) Was it easy to answer questions? | $3.19 \pm 0.89$ | $3.08 \pm 1.03$ | 0.2 |
| (Q5) Was the pace of the interaction with the robot appropriate? | $3.38 \pm 1.04$ | $3.08 \pm 1.07$ | **0.05** |
| (Q6) Did you know what you could say at each point? | $3.43 \pm 0.83$ | $3.24 \pm 1.02$ | 0.1 |
| (Q7) Was the robot fast and quick to reply to you? | $3.41 \pm 1.03$ | $3.46 \pm 1.04$ | 0.3 |
| (Q8) Did the robot work the way you expected it? | $3.46 \pm 0.87$ | $3.0 \pm 1.01$ | **0.008** |
| (Q9) Do you think you would use the robot in the future? | $3.27 \pm 1.19$ | $3.08 \pm 1.23$ | *n.s.* |
| **All** | $\mathbf{3.33 \pm 0.98}$ | $\mathbf{3.14 \pm 1.06}$ | **0.0002** |

## 7. RELATED WORK AND DISCUSSION

To position the work above in the literature, this section gives an overview of related work in two areas from a reinforcement learning perspective. First, we discuss the state-of-the-art in human-computer interaction with a particular focus on allowing flexible dialogue control and alternative methods to scale up to complex systems. We then discuss reinforcement learning applied to conversational robots, which surprisingly has received little attention to date. For all related work discussed, we highlight commonalities and differences with the approach proposed in this article.

### 7.1. Reinforcement Learning for Flexible and Scalable Dialogue Control

This section gives an overview of reinforcement learning for scalable and flexible dialogue control, which are the two main points we focus on in this article.

*Scalability.* Since human dialogue is a complex phenomenon, dialogue policies can quickly grow complex, especially for non-toy domains. Scalable learning methods are therefore an important research direction in dialogue policy learning. Two main solutions to this problem have been investigated, hierarchical decomposition and function approximation.

— Hierarchical reinforcement learning uses a divide-and-conquer approach to optimize sequential decision making problems with large state-action spaces [Barto and Mahadevan 2003]. This form of learning optimizes decision making at different levels of granularity, from high-level decisions,

such as how to combine verbal with non-verbal actions or when to switch to another subtask, to low-level actions such as when to apologize to a user, confirm or accept user contributions. [Cuayáhuitl 2009] presents an application of hierarchical RL to dialogue policy optimization for information-seeking dialogue in the travel domain. This work is based on model-free hierarchical learning [Dietterich 2000c]. Similarly, in [Cuayáhuitl and Dethlefs 2011] we present an application to dialogues in a spatial domain. We show that optimizing route planning and dialogue management jointly, the system learns to adapt to the user's prior knowledge of the environment, e.g. leading them past landmarks they are already familiar with or avoiding junctions where the user is likely to get confused. [Lemon 2011] claims to use hierarchical optimization to jointly optimize dialogue management with natural language generation for information presentation. In contrast to our work, he does not optimize subdialogues and therefore his approach is limited to small-domain systems. All of these approaches have benefitted from using hierarchical RL. Especially, the decomposition of behavior into subbehaviors leads to faster learning, reduces computational demands, provides opportunities for the reuse of subsolutions, and is more suitable for multi-task systems. The utility of hierarchical RL typically increases with the complexity and size of the state-action space of a system and has also been adopted for dialogue in combination with robust speech recognition [Pineau 2004] and natural language generation [Dethlefs and Cuayáhuitl 2010; Dethlefs 2013]. However, previous works in this area have ignored interaction flexibility.

— An alternative to a hierarchical decomposition approach are function approximation techniques. For example, Linear Function Approximation offers the advantage of generalization because dialogue policies can make decisions even for unseen dialogue states. A drawback typically lies in finding the right set of features to use for learning good policies. Another drawback is that no guarantees can be given that the learnt solutions will converge to an optimal policy. Previous related work has used hundreds or even thousands of features for dialogue policy learning from small data sets [Henderson et al. 2008] with feature selection [Li et al. 2009]. An alternative approach has proposed to optimize dialogue policies using a framework based on Kalman temporal differences in order to support linear and non-linear architectures [Daubigney et al. 2012]. Unfortunately, the authors do not report an evaluation with an end-to-end dialogue system with real users. Nonetheless, function approximation techniques have been appreciated for allowing agents to learn in a scalable way and achieve more robustness to unseen states.

In this article, we aim to bring the benefits of hierarchical learning and function approximation together in one framework in order to optimize dialogue systems with a repertoire of subdialogues.

*Flexibility.* The need for flexible dialogue control in conversational systems has long been recognized. For example, the limited behavior of dialogue systems designed with 'system-initiative' behavior has been extensively addressed by using 'mixed-initiative behavior' [Zue and Glass 2000; Bohus and Rudnicky 2009] in which the user is given a certain degree of control. Since hand-coding mixed-initiative behavior for complex systems is a daunting task, researchers have turned their attention to machine learning of dialogue strategies to support flexible and adaptive interactions. Reinforcement learning has again been a promising direction here [Lemon and Pietquin 2007], among other machine learning frameworks with practical application to interactive systems and robots [Cuayáhuitl et al. 2013]. Unfortunately, applications to complex dialogue systems (such as systems with multiple tasks, subtasks and modalities involving large state-action spaces) are not trivial due to scalability problems. Possible remedies have been discussed in the previous section, which have been applied in this article to achieve scalability and more flexible dialogue behavior.

Rigid dialogue behavior can also be a consequence of very small state representations [Walker 2000; Singh et al. 2002]. These have been appreciated because they prevent state spaces from growing too large (i.e. to avoid the curse of dimensionality) and also for their ability to lead to fast learning. On the other hand, compact tabular representations often cannot cover all possible situations in a dialogue, which can lead to unseen states. To address this problem, this article makes use of hierarchical controllers with function approximation and flexible transitions in the hierarchy.

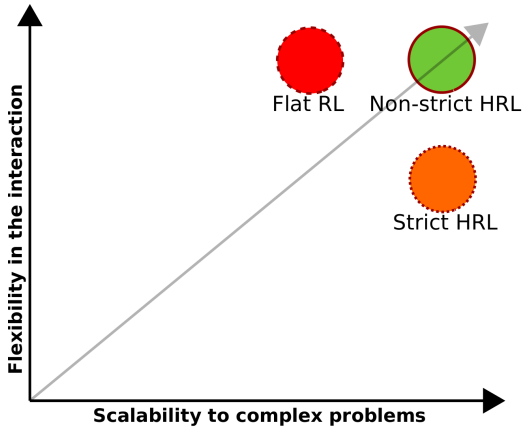



Fig. 9. Scalability vs. flexibility exhibited in flat Reinforcement Learning (RL) and Hierarchical Reinforcement Learning (HRL), where the more scalable and flexible the better.

*Trade-off Between Scalability and Flexibility.* Interactive systems that learn their behavior need to scale up to a large set of state variables and actions. This is especially true if systems have to provide support for multiple domains (or subdomains) rather than a single one. Hierarchical Reinforcement Learning (HRL) is attractive because it can learn quicker than flat RL (i.e. with less training data), but it may have less expressive power than flat RL due to the factorisation of the state-action space into subtasks. Therefore, while hierarchical RL is more scalable and less flexible, flat RL is less scalable and more flexible. Thus, the *trade-off* here consists in finding a mechanism to maintain scalability without sacrificing flexibility. Our proposed approach—based on non-strict HRL—described in Section 4 is our mechanism to handle this trade-off, see Figure 9. The following limitations are observed between these different forms of learning:

— Two key limitations of flat RL in contrast to HRL include scaling up to complex systems (e.g. a robot carrying out multiple activities in a house) and slow learning. The latter has been demonstrated by multiple authors [Sutton et al. 1999; Dietterich 2000a; Pineau 2004; Cuayáhuitl 2009]. The importance of HRL therefore grows according to the scale of the optimization problem.

— A notable limitation of strict HRL in contrast to flat RL is that the structure of interactions (hierarchy) is not known in advance and therefore it is difficult to specify. This may cause a mismatch between the interaction structures observed in the learning environment and those observed at deployment time—based on interaction with human users.

— A potential limitation of non-strict HRL in contrast to flat RL is the fact that the hierarchy and its corresponding elements may not be trivial to specify. Though once the hierarchy is specified, the non-strict HRL offers more scalability over flat RL. In addition, it offers more generalization to unseen transitions of subtasks than strict HRL, and therefore it deals better with the mismatch above faced by strict HRL. Thus, non-strict HRL aims to offer equivalent expressive power to flat RL but with the scalability properties of HRL.

— A noteworthy limitation of non-strict HRL in contrast to strict HRL are the additional elements to support flexible transitions across subtasks (i.e. subtask state transitions and dynamic state spaces), which require additional training. Nonetheless, the additional computational expense is justified by being able to train systems that exhibit more natural interactions.

Although our experiments above are based on a relatively easy optimization problem, our proposed approach can scale up to more complex problems with larger hierarchies. This is especially true if the concepts of modularity and abstraction are used in the hierarchy of subtasks (skills).

### 7.2. Reinforcement Learning for Conversational Robots

Reinforcement learning for human-robot interaction has pursued two main research directions: user modeling and multimodal dialogue policy learning. The methods discussed in this section have primarily focused on finding the best solution for the domain addressed.

— The first strand, user modeling, typically requires the robot to get to know their human interlocutor and adapt to them at several levels. [Mitsunaga et al. 2005] investigate interaction distances in human-robot interaction, which include behavioral features such as the personal distance from the interlocutor, gaze meeting, motion speed and timing. They argue that a robot should be able to recognize signs of discomfort in its human interlocutor, for example if they feel their personal space invaded, and adapt its behavior accordingly. To achieve this, they use direct estimation of the learnt policy without action-value functions and learn a policy for a particular cluster of users. [Atrash and Pineau 2009; Atrash et al. 2009] present work on adapting the conversational interface of an autonomous wheelchair to the preferences of its particular users. They investigate Bayesian reinforcement learning to train the interaction manager. In particular, they argue that maintaining posterior distributions of the state transitions and rewards can be useful to adapt the robot's policy to particular user preferences in a generalizable way. Whereas these previous investigations have optimized a single policy for the entire interaction, we optimize multiple policies (simultaneously) based on subdialogues. The latter is important to scale up not only to a large set of features in a single task but to optimizing interactions with multiple tasks or subtasks. Nonetheless, our proposed approach can be combined with such previous learning approaches in order unify their best properties.

— The second strand of work, multimodal policy learning, typically requires the robot to learn a mapping from multimodal states to actions. [Stiefelhagen et al. 2007] describe their developed technologies for a cooperative humanoid robot whose dialogue policy is trained using reinforcement learning. In [Cuayáhuitl and Kruijff-Korbayová 2011; Belpaeme et al. 2012] we describe a humanoid robot for long-term interaction whose multimodal dialogue policy is trained using hierarchical reinforcement learning. Recent work on social robots makes use of MDP-based policies to specify their behavior from audio-visual inputs of multiple conversants [Keizer et al. 2013]. These research teams train their robots in simulation and come to the shared conclusion that the seamless integration of trained skills with different components is a challenge on its own.

Similarly to non-conversational robots, who typically focus on interaction with their physical environment (see [Kober et al. 2013] for an introduction to reinforcement learning in robotics), conversational robots also face the challenge of learning in unknown and unstructured environments involving unknown people, unknown objects, and unknown situations. This requires continuous interpretation and reaction of incoming user and environment signals and demands a tight integration of multiple multimodal input and output channels, such as coordinating incoming visual and verbal signals with planning gesture and language output. This paper makes a contribution to more flexible and scalable human-robot interaction by extending and evaluating an RL approach previously investigated in dialogue systems to human-robot interaction. Also, we present an interactive robot able to deal with less-rigid interactions and adapt to the dialogue structure of users.

## 8. CONCLUSION AND FUTURE WORK

The contribution of this article is a novel approach for optimizing the behavior of interactive systems and robots by extending an existing hierarchical learning algorithm to support non-strict hierarchical control and policies represented with function approximation. In our approach, we relax strict hierarchical control by allowing users to navigate more flexibly across the available subdialogues. To achieve that we extend each reinforcement learning agent in the hierarchy with a subtask transition function and a dynamic state space. We evaluated our approach by incorporating it into a robot dialogue system that learns to play quiz games. Our experimental results, based on simulations trained from Wizard-of-Oz data and experiments with human users, show that our approach is promising. It

leads to more flexible interactions than a policy that uses strict control and it is preferred by human users. We therefore argue that our approach represents an important step forward in the development of trainable dialogue systems that combine the benefits of scalability and flexible interaction and allow the user to take more initiative. This approach aims to be of practical use for training RL-based and real world dialogue systems rather than toy systems.

We suggest the following research avenues for RL-based interactive systems and robots:

(1) Investigate further how interactive systems and robots can induce the characterization of their environment in a more automatic way, including their state and action space, transition function and reward function. For example, one could investigate how to discover the interaction structure (hierarchy) and the reward function throughout the interaction. A method for hierarchy discovery is described in [Mehta et al. 2008].
(2) Investigate when to relearn interaction policies. In this article, we assumed that a policy prelearnt using simulations can be used for interaction with real users. However, due to the dynamic nature of the learning environment, it remains to be defined when the policy should be relearnt and adapted to new unseen behavior. Continuous monitoring of a new policy in comparison with a previously learned policy is required in order to improve the agent's performance.
(3) Learn subtask transition functions in a way that the system can suggest how to navigate in the hierarchy of subdialogues. In our system, the user was allowed to navigate flexibly across subdialogues. However, the robot could learn such kinds of patterns for navigating across subdialogues as well in order to exhibit more adaptive behavior itself. This feature may reduce monotonicity in the interaction and could potentially increase user satisfaction.
(4) Optimize dialogue control combining verbal and non-verbal behaviors from human demonstrations. Our humanoid robot used dialogue policies to control the main flow of the interaction, and a deterministic mapping from dialogue actions to gestures. For example, [Cuayáhuitl and Dethlefs 2012] use multiagent reinforcement learning to coordinate verbal and non-verbal actions. Similarly, gestures can be jointly optimized with natural language generation to exhibit more natural multimodal output. For other kinds of joint optimizations, see [Dethlefs et al. 2012b; Dethlefs and Cuayáhuitl 2011b; 2011a; Lemon 2011].
(5) Optimize dialogue control by adding linguistically-motivated phenomena such as multimodal turn-taking [Chao and Thomaz 2012; Nalin et al. 2012] and incremental processing [Schlangen and Skantze 2009; Dethlefs et al. 2012a; Cuayáhuitl et al. 2013]. These features have shown to be important contributors to improve performance and enhance naturalness in the interactions.
(6) Investigate spatially-aware interactive robots. Our robot was not aware of the spatial environment in which it interacted with users. This requires a more tight integration with other spatially-aware components [Cuayáhuitl and Dethlefs 2011; Cuayáhuitl and Dethlefs 2011; Janarthanam et al. 2012] and also reasoning about the spatial environment [Frommberger 2012].
(7) Investigate our proposed approach within an open-ended domain using a large knowledge base [Betteridge et al. 2009]. All of these research avenues invite to unify multiple forms of machine learning such as (multi-agent) reinforcement learning (from demonstration) combined with supervised/unsupervised/semi-supervised/active learning, see [Cuayáhuitl et al. 2013].
(8) Last but not least, in this article we demonstrated the effectiveness of non-strict HRL experimentally. It remains to be demonstrated from theoretical perspectives, e.g. convergence guarantees.

## ACKNOWLEDGMENTS

This work was supported by the European FP7 research projects ALIZ-E (ICT-248116), PARLANCE (287615), and SPACEBOOK (270019). We would like to thank the following people for helping to test, debug, and improve the integrated robot system used in this article: Georgios Athanasopoulos, Ilaria Baroni, Paul Baxter, Aryel Beck, Tony Belpaeme, Bert Bierman, Lola Caamero, Mariona Coll, Piero Cosi, Yannis Demiris, Valentin Enescu, Alessio Giusti, Antoine Hiolle, Remi Humbert, Bernd Kiefer, Boris Kozlov, Rosmarijn Looije, Marco Nalin, Mark Neerincx, Giulio Paci, Stefania Racioppa, Raquel Ros, Marc Schröder, Giacomo Sommavilla, Fabio Tesser. We also would like to thank Mariona Coll for helping to carry out the evaluation with real users. Finally, we thank our reviewers for helping to improve the clarity of this article.

## A. STATE SPACE

This section shows the initial state space used by our reinforcement learning agents (before growing due to unseen states). It is based on the context-free grammar that defines the initial state space for each agent in the hierarchy of subtasks: *root*, *RobotAsks* and *UserAsks*; respectively. Notation: $\wedge$ means *and*, $\vee$ means *or*, and \$ is a non-terminal symbol that expands non-terminal and terminal symbols. The state space of each learning agent corresponds to the full expansion of its non-terminal symbols, which corresponds to (nested) conjunctions of the form `StateVariable(value)`.

$\langle L \rangle$ ::= $\langle (Quiz(root)) \rangle \wedge \langle \$root) \rangle \vee \langle (Quiz(RobotAsks)) \rangle \wedge \langle \$subgame) \rangle \vee \langle (Quiz(UserAsks)) \rangle \wedge \langle \$subgame) \rangle$

$\langle \$root \rangle ::= \langle Salutation(\$noneclosing) \rangle$

$\langle \$root \rangle ::= \langle Salutation(greeting) \rangle \wedge \langle UserName(\$knownunknown) \rangle$

$\langle \$root \rangle ::= \langle Salutation(greeting) \rangle \wedge \langle UserName(filled,ConfScore(\$score)) \rangle$

$\langle \$root \rangle ::= \langle Salutation(greeting) \rangle \wedge \langle UserName(filled,ConfScore(\$score)) \rangle \wedge \langle Confirmed(\$yesno) \rangle$

$\langle \$root \rangle ::= \langle Salutation(withName) \rangle \wedge \langle UserName(known) \rangle$

$\langle \$root \rangle ::= \langle \$userKnownAndGreeted \rangle \wedge \langle PlayGame(no,ConfScore(\$score)) \rangle$

$\langle \$root \rangle ::= \langle \$userKnownAndGreeted \rangle \wedge \langle PlayGame(ready) \rangle \wedge \langle GameOver(yes) \rangle$

$\langle \$root \rangle ::= \langle \$userKnownAndGreeted \rangle \wedge \langle PlayGame(ready) \rangle \wedge \langle Instructions(unprovided) \rangle$

$\langle \$root \rangle ::= \langle \$userKnownAndGreeted \rangle \wedge \langle PlayGame(unknown) \rangle$

$\langle \$root \rangle ::= \langle \$userKnownAndGreeted \rangle \wedge \langle PlayGame(yes,ConfScore(\$score)) \rangle$

$\langle \$root \rangle ::= \langle \$userKnownAndGreeted \rangle \wedge \langle \$userReadyAndInstructed \rangle \wedge \langle Asker(\$robotuser,ConfScore(\$score)) \rangle$

$\langle \$root \rangle ::= \langle \$userKnownAndGreeted \rangle \wedge \langle \$userReadyAndInstructed \rangle \wedge \langle Asker(\$robotuser) \rangle \wedge \langle QuizGame(\$gamestatus) \rangle$

$\langle \$root \rangle ::= \langle \$userKnownAndGreeted \rangle \wedge \langle \$userReadyAndInstructed \rangle \wedge \langle Asker(\$robotuser) \rangle \wedge \langle QuizGame(\$stopkeepplaying,ConfScore(\$score)) \rangle$

$\langle \$root \rangle ::= \langle \$userKnownAndGreeted \rangle \wedge \langle \$userReadyAndInstructed \rangle \wedge \langle Asker(unknown) \rangle$

$\langle \$root \rangle ::= \langle \$userKnownAndGreeted \rangle \wedge \langle \$userReadyAndInstructed \rangle \wedge \langle QuizGame(stopPlaying) \rangle \wedge \langle GlobalGameScore(unprovided) \rangle$

$\langle \$root \rangle ::= \langle \$userKnownAndGreeted \rangle \wedge \langle \$userReadyAndInstructed \rangle \wedge \langle QuizGame(stopPlaying) \rangle \wedge \langle GlobalGameScore(semiprovided) \rangle$

$\langle \$root \rangle ::= \langle \$userKnownAndGreeted \rangle \wedge \langle \$userReadyAndInstructed \rangle \wedge \langle \$stopGameAndScoreProvided \rangle \wedge \langle GameFun(unknown) \rangle$

$\langle \$root \rangle ::= \langle \$userKnownAndGreeted \rangle \wedge \langle \$userReadyAndInstructed \rangle \wedge \langle \$stopGameAndScoreProvided \rangle \wedge \langle GameFun(\$yesno,ConfScore(\$score)) \rangle$

$\langle \$subgame \rangle ::= \langle GameInstructions(provided) \rangle \wedge \langle PlayGame(\$yesno,ConfScore(\$score)) \rangle$

$\langle \$subgame \rangle ::= \langle GameInstructions(provided) \rangle \wedge \langle PlayGame(unknown) \rangle$

$\langle \$subgame \rangle ::= \langle GameInstructions(unprovided) \rangle$

$\langle \$subgame \rangle ::= \langle \$userReadyToPlay \rangle \wedge \langle GameOver(yes) \rangle$

$\langle \$subgame \rangle ::= \langle \$userReadyToPlay \rangle \wedge \langle QuestionState(\$question) \rangle$

$\langle \$subgame \rangle ::= \langle \$userReadyToPlay \rangle \wedge \langle QuestionState(askedWithAnswers) \rangle \wedge \langle AnswerState(unclassified,ConfScore(\$score)) \rangle$

$\langle \$subgame \rangle ::= \langle \$userReadyToPlay \rangle \wedge \langle QuestionState(askedWithAnswers) \rangle \wedge \langle AnswerState(\$correctincorrect) \rangle$

$\langle \$subgame \rangle ::= \langle \$userReadyToPlay \rangle \wedge \langle QuestionState(askedWithAnswers) \rangle \wedge \langle AnswerState(\$unknownunanswered) \rangle$

$\langle \$subgame \rangle ::= \langle \$userReadyToPlay \rangle \wedge \langle QuestionState(confirmed) \rangle \wedge \langle MaxQuestions(\$yesno) \rangle$

$\langle \$subgame \rangle ::= \langle \$userReadyToPlay \rangle \wedge \langle QuestionState(confirmed) \rangle \wedge \langle MaxQuestions(yes) \rangle \wedge \langle GameScore(unknown) \rangle$

$\langle \$subgame \rangle ::= \langle \$userReadyToPlay \rangle \wedge \langle QuestionState(confirmed) \rangle \wedge \langle MaxQuestions(yes) \rangle \wedge \langle \$gameScoreKnownAndExpressed \rangle$

$\langle \$subgame \rangle ::= \langle \$userReadyToPlay \rangle \wedge \langle QuestionState(reaskedWithAnswers) \rangle \wedge \langle AnswerState(\$answer) \rangle$

$\langle \$subgame \rangle ::= \langle \$userReadyToPlay \rangle \wedge \langle QuestionState(reaskedWithAnswers) \rangle \wedge \langle AnswerState(unclassified,ConfScore(\$score)) \rangle$

$\langle \$stopGameAndScoreProvided \rangle ::= \langle QuizGame(stopPlaying) \rangle \wedge \langle GlobalGameScore(provided) \rangle$

$\langle \$userKnownAndGreeted \rangle ::= \langle Salutation(regreeting) \rangle \wedge \langle UserName(known) \rangle$

$\langle \$userReadyAndInstructed \rangle ::= \langle PlayGame(ready) \rangle \wedge \langle Instructions(provided) \rangle$

$\langle \$userReadyToPlay \rangle ::= \langle GameInstructions(provided) \rangle \wedge \langle PlayGame(ready) \rangle$
$\langle \$answer \rangle ::= \langle unanswered \rangle \mid \langle \$correctincorrect \rangle \mid \langle unknown \rangle \mid \langle revealed \rangle$
$\langle \$correctincorrect \rangle ::= \langle correct \rangle \mid \langle incorrect \rangle$
$\langle \$gamestatus \rangle ::= \langle unplayed \rangle \mid \langle playing \rangle \mid \langle semiplayed \rangle \mid \langle played \rangle \mid \langle interrupted \rangle$
$\langle \$gameScoreKnownAndExpressed \rangle ::= \langle GameScore(\$goodbad) \rangle \wedge \langle ExpressedScore(\$yesno) \rangle$
$\langle \$goodbad \rangle ::= \langle good \rangle \mid \langle bad \rangle$
$\langle \$knownunknown \rangle ::= \langle known \rangle \mid \langle unknown \rangle$
$\langle \$noneclosing \rangle ::= \langle none \rangle \mid \langle closing \rangle$
$\langle \$question \rangle ::= \langle \$questionasked \rangle \mid \langle \$questionheard \rangle \mid \langle \$questionstatus \rangle$
$\langle \$questionasked \rangle ::= \langle askedWithUnknownQuestion \rangle \mid \langle reaskedWithoutAcknowledgement \rangle$
$\langle \$questionheard \rangle ::= \langle askedButHeardBefore \rangle \mid \langle askedButUnheardBefore \rangle$
$\langle \$questionstatus \rangle ::= \langle unknown \rangle \mid \langle unasked \rangle \mid \langle askedWithoutAnswers \rangle \mid \langle reaskedWithoutAnswers \rangle$
$\langle \$robotuser \rangle ::= \langle robot \rangle \mid \langle user \rangle$
$\langle \$score \rangle ::= \langle 0.1 \rangle \mid \langle 0.2 \rangle \mid \langle 0.3 \rangle \mid \langle 0.4 \rangle \mid \langle 0.5 \rangle \mid \langle 0.6 \rangle \mid \langle 0.7 \rangle \mid \langle 0.8 \rangle \mid \langle 0.9 \rangle \mid \langle 1.0 \rangle$
$\langle \$stopkeepplaying \rangle ::= \langle stopPlaying \rangle \mid \langle keepPlaying \rangle$
$\langle \$unknownunanswered \rangle ::= \langle unknown \rangle \mid \langle unanswered \rangle$
$\langle \$yesno \rangle ::= \langle yes \rangle \mid \langle no \rangle$

## REFERENCES


A. Atrash, R. Kaplow, J. Villemure, R. West, H. Yamani, and J. Pineau. 2009. Development and Validation of a Robust Speech Interface for Improved Human-Robot Interaction. *International Journal of Social Robotics* 1, 4 (2009), 345–356.

A. Atrash and J. Pineau. 2009. A Bayesian Reinforcement Learning Approach for Customizing Human-Robot Interfaces. In *International Conference on Intelligent User Interfaces (IUI)*. 355–360.

J. Baillie. 2005. URBI: Towards a Universal Robotic Low-Level Programming Language. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 3219–3224.

A. Barto and S. Mahadevan. 2003. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems: Theory and Applications* 13, 1-2 (2003), 41–77.

A. Beck, L. Cañamero, and K.A. Bard. 2010. Towards an Affect Space for Robots to Display Emotional Body Language. In *International Symposium on Robot and Human Interactive Communication (Ro-Man)*. IEEE, 464–469.

T. Belpaeme, P. Baxter, R. Read, R. Wood, H. Cuayáhuitl, B. Kiefer, S. Racioppa, I. Kruijff Korbayová, G. Athanasopoulos, V. Enescu, R. Looije, M. Neerincx, Y. Demiris, R. Ros-Espinoza, A. Beck, L. Canãmero, A. Hiolle, M. Lewis, I. Baroni, M. Nalin, P. Cosi, G. Paci, F. Tesser, G. Sommavilla, Humbert., and R. 2012. Multimodal Child-Robot Interaction: Building Social Bonds. *Journal of Human-Robot Interaction* 1, 2 (2012), 32 – 455.

J. Betteridge, A. Carlson, S. A. Hong, E. R. Jr. Hruschka, E. L. M. Law, T. M. Mitchell, and S. H. Wang. 2009. Toward Never Ending Language Learning. In *AAAI Spring Symposium: Learning by Reading and Learning to Read*. 1–2.

Dan Bohus and Alexander I. Rudnicky. 2009. The RavenClaw dialog management framework: Architecture and systems. *Computer Speech & Language* 23, 3 (2009), 332–361.

F. Cao and S. Ray. 2012. Bayesian Hierarchical Reinforcement Learning. In *Neural Information Processing Systems Foundation (NIPS)*. 73–81.

C. Chao and A. L. Thomaz. 2012. Timing in Multimodal Reciprocal Interactions: Control and Analysis Using Timed Petri Nets. *Journal of Human-Robot Interaction* 1, 1 (2012), 4–25.

P. A. Crook, A. Wang, X. Liu, and L. Lemon. 2012. A Statistical Spoken Dialogue System using Complex User Goals and Value Directed Compression. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*. 46–50.

H. Cuayáhuitl. 2009. *Hierarchical Reinforcement Learning for Spoken Dialogue Systems*. Ph.D. Dissertation. School of Informatics, University of Edinburgh.

H. Cuayáhuitl and N. Dethlefs. 2011. Optimizing Situated Dialogue Management in Unknown Environments. In *Annual Conference of the International Speech Communication Association (INTERSPEECH)*. Florence, Italy, 1009–1012.

H. Cuayáhuitl and N. Dethlefs. 2011. Spatially-Aware Dialogue Control Using Hierarchical Reinforcement Learning. *ACM Transactions on Speech and Language Processing* 7(3) (2011), 5:1–5:26.

H. Cuayáhuitl and N. Dethlefs. 2012. Hierarchical Multiagent Reinforcement Learning for Coordinating Verbal and Nonverbal Actions in Robots. In *ECAI Workshop on Machine Learning for Interactive Systems (MLIS)*. Montpellier, France, 27–29.

H. Cuayáhuitl, N. Dethlefs, H. Hastie, and O. Lemon. 2013. Barge-in Effects in Bayesian Dialogue Act Recognition and Simulation. In *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*.

H. Cuayáhuitl and I. Kruijff-Korbayová. 2011. Learning Human-Robot Dialogue Policies Combining Speech and Visual Beliefs. In *International Workshop on Spoken Dialogue Systems (IWSDS)*. 133–140.

H. Cuayáhuitl, S. Renals, O. Lemon, and H. Shimodaira. 2007. Hierarchical Dialogue Optimization Using Semi-Markov Decision Processes. In *Annual Conference of the International Speech Communication Association (INTERSPEECH)*. Antwerp, Belgium, 2693–2696.

H. Cuayáhuitl, S. Renals, O. Lemon, and H. Shimodaira. 2010. Evaluation of a Hierarchical Reinforcement Learning Spoken Dialogue System. *Computer Speech and Language* 24, 2 (2010), 395–429.

H. Cuayáhuitl, M. van Otterlo, N. Dethlefs, and L. Frommberger. 2013. Machine Learning for Interactive Systems and Robots: A Brief Introduction. In *IJCAI Workshop on Machine Learning for Interactive Systems (MLIS)*.

L. Daubigney, M. Geist, S. Chandramohan, and O. Pietquin. 2012. A Comprehensive Reinforcement Learning Framework for Dialogue Management Optimization. *Journal of Selected Topics in Signal Processing* 6, 8 (2012).

N. Dethlefs. 2013. *Hierarchical Joint Learning for Natural Language Generation*. Ph.D. Dissertation. University of Bremen.

N. Dethlefs and H. Cuayáhuitl. 2010. Hierarchical Reinforcement Learning for Adaptive Text Generation. In *International Conference on Natural Language Generation (INLG)*. Dublin, Ireland.

N. Dethlefs and H. Cuayáhuitl. 2011a. Combining Hierarchical Reinforcement Learning and Bayesian Networks for Natural Language Generation in Situated Dialogue. In *European Workshop on Natural Language Generation (ENLG)*. Nancy, France, 110–120.

N. Dethlefs and H. Cuayáhuitl. 2011b. Hierarchical Reinforcement Learning and Hidden Markov Models for Task-Oriented Natural Language Generation. In *Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT)*. Portland, OR, USA, 654–659.

N. Dethlefs, H. Cuayáhuitl, and J. Viethen. 2011. Optimising Natural Language Generation Decision Making for Situated Dialogue. In *Annual Meeting on Discourse and Dialogue (SIGdial)*. Portland, Oregon, USA.

N. Dethlefs, H. Hastie, R. Rieser, and O. Lemon. 2012a. Optimising Incremental Dialogue Decisions Using Information Density for Interactive Systems. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Jeju, South Korea.

N. Dethlefs, V. Rieser, H. Hastie, and O. Lemon. 2012b. Towards Optimising Modality Allocation for Multimodal Output Generation in Incremental Dialogue. In *ECAI Workshop on Machine Learning for Interactive Systems (MLIS)*. Montpellier, France, 31–36.

T. Dietterich. 2000a. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research* 13, 1 (2000), 227–303.

T. Dietterich. 2000b. An Overview of MAXQ Hierarchical Reinforcement Learning. In *Symposium on Abstraction, Reformulation, and Approximation (SARA)*. 26–44.

T. G. Dietterich. 2000c. An Overview of MAXQ Hierarchical Reinforcement Learning. In *Symposium on Abstraction, Reformulation, and Approximation (SARA)*. HorseShoeBay, Texas, USA.

L. Frommberger. 2012. *Qualitative Spatial Abstraction in Reinforcement Learning*. Springer-Verlag New York.

M. Gašić and S. Young. 2011. Effective Handling of Dialogue State in the Hidden Information State POMDP-based Dialogue Manager. *ACM Transactions on Speech and Language Processing* 7(3) (2011).

P. Heeman. 2007. Combining Reinforcement Learning with Information-State Update Rules. In *Human Language Technology Conference (HLT)*. Rochester, NY, USA, 268–275.

J. Henderson, O. Lemon, and K. Georgila. 2008. Hybrid Reinforcement/Supervised Learning of Dialogue Policies from Fixed Data Sets. *Computational Linguistics* 34, 4 (2008), 487–511.

S. Janarthanam, L. Lemon, X. Liu, P. Bartie, W. Mackaness, T. Dalmas, and J. Goetze. 2012. Integrating Location, Visibility, and Question-Answering in a Spoken Dialogue System for Pedestrian City Exploration. In *Workshop on Semantics and Pragmatics of Dialogue (SEMDIAL)*. Paris, France.

F. Jurčíček, B. Thomson, and S. Young. 2011. Natural Actor and Belief Critic: Reinforcement Algorithm for Learning Parameters of Dialogue Systems Modelled as POMDPs. *ACM Transactions on Speech and Language Processing* 7, 3 (2011).

S. Keizer, M.E. Foster, O. Lemon, A. Gaschler, and Giuliani. M. 2013. Training and evaluation of an MDP model for social multi-user human-robot interaction. In *Annual Meeting on Discourse and Dialogue (SIGDIAL)*.

J. Kober, J. A. Bagnell, and J. Peters. 2013. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research* 32, 11 (2013).

I. Kruijff-Korbayová, H. Cuayáhuitl, B. Kiefer, M. Schröder, P. Cosi, G. Paci, G. Sommavilla, F. Tesser, H. Sahli, G. Athanasopoulos, W. Wang, V. Enescu, and W. Verhelst. 2012b. A Conversational System for Multi-Session Child-Robot Interaction with Several Games. In *German Conference on Artificial Intelligence (KI)*. system demonstration description.

I. Kruijff-Korbayová, H. Cuayáhuitl, B. Kiefer, M. Schröder, P. Cosi, G. Paci, G. Sommavilla, F. Tesser, H. Sahli, G. Athanasopoulos, W. Wang, V. Enescu, and W. Verhelst. 2012a. Spoken Language Processing in a Conversational System for Child-Robot Interaction. In *INTERSPEECH Workshop on Child-Computer Interaction*.

O. Lemon. 2011. Learning What to Say and How to Say It: Joint Optimization of Spoken Dialogue Management and Natural Language Generation. *Computer Speech and Language* (2011).

O. Lemon and O. Pietquin. 2007. Machine Learning for Spoken Dialogue Systems. In *Annual Conference of the International Speech Communication Association (INTERSPEECH)*. Antwerpen, Belgium, 2685–2688.

E. Levin, R. Pieraccini, and W. Eckert. 2000. A Stochastic Model of Human Machine Interaction for Learning Dialog Strategies. *IEEE Transactions on Speech and Audio Processing* 8, 1 (2000), 11–23.

L. Li, D. J. Williams, and S. Balakrishnan. 2009. Reinforcement Learning for Dialog Management using Least-Squares Policy Iteration and Fast Feature Selection. In *Annual Conference of the International Speech Communication Association (INTERSPEECH)*. Brighton, United Kingdom, 2475–2478.

D. Litman, M. Kearns, S. Singh, and M. Walker. 2000. Automatic Optimization of Dialogue Management. In *International Conference on Computational Linguistics (COLING)*. Saarbrucken, Germany, 502–508.

N. Mehta, S. Ray, P. Tadepalli, and T. G. Dietterich. 2008. Automatic discovery and transfer of MAXQ hierarchies. In *International Conference on Machine Learning (ICML)*. 648–655.

N. Mitsunaga, C. Smith, T. Kanda, H. Ishiguro, and N. Hagita. 2005. Robot Behavior Adaptation for Human-Robot Interaction Based on Policy Gradient Reinforcement Learning. In *International Conference on Intelligent Robots and Systems (IROS)*. 218–225.

M. Nalin, I. Baroni, I. Kruijff-Korbayová, L. Cañamero, M. Lewis, A. Beck, H. Cuayáhuitl, and A. Sanna. 2012. Children's Adaptation in Multi-Session Interaction with a Humanoid Robot. In *International Symposium on Robot and Human Interactive Communication (RO-MAN)*. 351–357.

O. Pietquin. 2011. Batch Reinforcement Learning for Spoken Dialogue Systems with Sparse Value Function Approximation. In *NIPS Workshop on Learning and Planning from Batch Time Series Data*. Vancouver, Canada.

O. Pietquin, M. Geist, and S. Chandramohan. 2011. Sample-Efficient Batch Reinforcement Learning for Dialogue Management Optimization. *ACM Transactions on Speech and Language Processing* 7, 3 (2011), 7.

J. Pineau. 2004. *Tractable Planning Under Uncertainty: Exploiting Structure*. Ph.D. Dissertation. Carnegie Mellon University.

N. Roy, J. Pineau, and S. Thrun. 2000. Spoken Dialogue Management Using Probabilistic Reasoning. In *International Conference on Computational Linguistics (ACL)*. Hong Kong, 93–100.

D. Schlangen and G. Skantze. 2009. A General, Abstract Model of Incremental Dialogue Processing. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*. Athens, Greece.

S. Singh, D. Litman, M. Kearns, and M. Walker. 2002. Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System. *Journal of Artificial Intelligence Research* 16 (2002), 105–133.

R. Stiefelhagen, H. K. Ekenel, C. Fügen, P. Gieselmann, H. Holzapfel, F. Kraft, K. Nickel, M. Voit, and A. Waibel. 2007. Enabling Multimodal Human-Robot Interaction for the Karlsruhe Humanoid Robot. *IEEE Transactions on Robotics* 23, 5 (2007), 840–851.

R. Sutton and A. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Richard S. Sutton, Doina Precup, and Satinder P. Singh. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence* 112, 1-2 (1999), 181–211.

C. Szepesvári. 2010. *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers.

A. L. Thomaz and C. Breazeal. 2006. Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance. In *AAAI Conference on Artificial Intelligence*. 1000–1006.

B. Thomson. 2009. *Statistical Methods for Spoken Dialogue Management*. Ph.D. Dissertation. University of Cambridge.

M. Walker. 2000. An Application of Reinforcement Learning to Dialogue Strategy Selection in a Spoken Dialogue System for Email. *Journal of Artificial Intelligence Research* 12 (2000), 387–416.

J. Williams. 2007. Partially Observable Markov Decision Processes for Spoken Dialog Systems. *Computer Speech and Language* 21, 2 (2007), 393–422.

J. Williams. 2008. The Best of Both Worlds: Unifying Conventional Dialog Systems and POMDPs. In *Annual Conference of the International Speech Communication Association (INTERSPEECH)*. Brisbane, Australia.

S. Young. 2000. Probabilistic Methods in Spoken Dialogue Systems. *Philosophical Transactions of the Royal Society (Series A)* 358, 1769 (2000), 1389–1402.

Y. Young, M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, Thomson B., and K. Yu. 2010. The Hidden Information State Model: A Practical Framework for POMDP-based Spoken Dialogue Management. *Computer Speech and Language* 24, 2 (2010), 150–174.

V. Zue and J. Glass. 2000. Conversational Interfaces: Advances and Challenges. *IEEE Transactions on Speech and Audio Processing* 88, 8 (2000), 1166–1180.